Mendelova univerzita v Brně
Provozně ekonomická fakulta

Redakční systém s integrací webových služeb

Diplomová práce

Vedoucí práce:
Ing. Ondřej Popelka, Ph.D.

Bc. Roman Pospěch, DiS.

Brno 2017

Děkuji panu Ing. Ondřeji Popelkovi, Ph.D. za odborné vedení této práce, cenné rady a vstřícný přístup. Mé poděkování patří také mojí rodině a přítelkyni za podporu během celého studia.

Čestné prohlášení

Prohlašuji, že jsem tuto práci: **Redakční systém s integrací webových služeb** vypracoval samostatně a veškeré použité prameny a informace jsou uvedeny v seznamu použité literatury. Souhlasím, aby moje práce byla zveřejněna v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů, a v souladu s platnou *Směrnicí o zveřejňování vysokoškolských závěrečných prací.*

Jsem si vědom, že se na moji práci vztahuje zákon č. 121/2000 Sb., autorský zákon, a že Mendelova univerzita v Brně má právo na uzavření licenční smlouvy a užití této práce jako školního díla podle § 60 odst. 1 Autorského zákona.

Dále se zavazuji, že před sepsáním licenční smlouvy o využití díla jinou osobou (subjektem) si vyžádám písemné stanovisko univerzity o tom, že předmětná licenční smlouva není v rozporu s oprávněnými zájmy univerzity, a zavazuji se uhradit případný příspěvek na úhradu nákladů spojených se vznikem díla, a to až do jejich skutečné výše.

V Brně dne 16. května 2017 .....

Abstract

Pospěch, Roman. *Content management system with web services integration.* Brno, 2017.

This thesis deals with the development of content management system that allows users to easily manage their websites. In order to facilitate the user's work, the issue of web services integration is also addressed. Based on the specified requirements, commonly used systems are compared in terms of the simplicity of web administration and the ability to connect to various web services. According to these findings, it is designed and implemented content management system that allows the user to perform operations with web services provided by Facebook, Google and Twitter. The implemented system was deployed and tested on the website that was created according to real requirements.

Keywords

Web application, content management system, web services integration, REST API, Nette Framework.

Abstrakt

Pospěch, Roman. *Redakční systém s integrací webových služeb.* Brno, 2017.

Tato práce se zabývá vývojem redakčního systému, který uživateli umožní snadnou správu webových stránek. Za účelem usnadnění práce uživateli je řešena také problematika integrace webových služeb. Na základě specifikovaných požadavků jsou porovnány běžně používané systémy z hlediska jednoduchosti správy webu a možnosti propojení s různými webovými službami. Podle těchto poznatků je navržen a implementován redakční systém, který umožňuje provádět operace s webovými službami společností Facebook, Google a Twitter. V závěru práce byl systém nasazen a otestován na webu, který byl vytvořen podle reálných požadavků.

Klíčová slova

Webová aplikace, redakční systém, integrace webových služeb, REST API, Nette Framework.

Obsah

1 Úvod a cíl práce

1.1 Úvod

Pokud chce jedinec nebo organizace vytvořit web, na kterém bude prezentovat sebe nebo službu či produkt, který nabízí, může postupovat různými způsoby. Větší organizace zpravidla pověří tvorbou a správou svých webových stránek agenturu. Někteří jednotlivci nebo menší a nevýdělečné organizace zase raději využijí některý z běžně používaných redakčních systémů, ve kterém si web vytvoří (či nechají vytvořit) a spravují jej pak zcela sami.

Redakčních systémů existuje poměrně velké množství. S jejich pomocí je možné vytvořit i relativně komplexní webové prezentace. V málokterém z nich však uživatel docílí požadovaného výsledku bez pokročilých uživatelských znalostí. Lze říci, že čím více možností redakční systém poskytuje, tím náročnější je na pochopení a ovládání.

Pro řadu uživatelů ovšem představuje používání takto komplexních systémů značný problém. Nechtějí trávit čas mnohdy složitým nastavováním struktury a vzhledu svých webových stránek. Chtějí se věnovat pouze tvorbě jejich obsahu, a to co nejvíce přímočarou cestou. Namísto komplexního nástroje pro tvorbu a správu webů, kterým běžně používané redakční systémy jsou, tedy potřebují systém vytvořený na míru jejich požadavkům. Pověří tedy někoho jiného, aby pro ně takový systém vytvořil.

1.2 Cíl práce

Cílem práce je vytvořit redakční systém, který umožní efektivní správu obsahu. Tento systém by měl být snadno upravitelný a rozšiřitelný ze strany tvůrce individuálních webových stránek. Koncový uživatel pak obdrží aplikaci vytvořenou na míru, kde bude odstíněn od složitých možností nastavení. Takto vytvořená aplikace tedy bude obsahovat vždy jen ty komponenty, které uživatel vyžaduje a spravovat bude možné právě ten obsah, který uživatel potřebuje frekventovaně měnit.

Aby byla koncovému uživateli usnadněna práce, budou do systému integrované různé webové služby. Jako jeden z příkladů výhod této integrace lze uvést situaci, kdy uživatel na svých stránkách vytvoří událost. Ta pak může být automaticky vložena do Google kalendáře uživatele, její poloha se objeví spolu s ostatními událostmi na Google Maps a informace o vzniku této události může být publikována na Facebooku, Twitteru nebo Google+.

Vytvořený systém také umožní přístup k operacím s daty v tomto systému prostřednictvím vlastního API. Pro tvorbu tohoto rozhraní bude využita architektura REST.

2 Specifikace požadavků

Cílem této kapitoly je co nejpřesněji vymezit, jaké vlastnosti a funkce má vytvářená aplikace nabízet. Před samotnou specifikací požadavků je ale třeba vysvětlit význam některých pojmů, které se v následujícím textu budou vyskytovat:

- **Koncový uživatel systému** – jedinec nebo skupina či organizace, která bude redakční systém používat. Jde o zadavatele konkrétních požadavků na vlastnosti systému.
- **Instance systému** – kopie kódu redakčního systému. Každý koncový uživatel obdrží vlastní instanci systému, která bude vyhovovat jeho požadavkům.
- **Implementátor systému** – jedinec nebo skupina či organizace, která je zodpovědná za vytváření, nasazování a údržbu instancí redakčního systému.

2.1 Funkční požadavky

Výčet požadavků na funkcionalitu redakčního systému:

- Redakční systém musí umožnit vytváření, úpravu a odstraňování jednotlivých stránek webu.
- Redakční systém musí umožnit koncovému uživateli vytváření, zobrazení, úpravu a odstraňování různých typů položek. Těmi můžou být například články, události, fotoalba apod. Umožněné operace se však budou odvíjet od požadavků koncového uživatele systému.
- U každého typu položky musí být možné vkládat komentáře a sdílet tyto položky na sociálních sítích. Umožnění této funkcionality bude ovšem opět závislé na požadavcích koncového uživatele.
- Systém umožní provádění CRUD operací se zmíněnými položkami také přes vlastní REST API.
- Systém musí umožnit vytváření, úpravu a odstraňování či blokování uživatelských účtů v dané instanci systému.
- Systém musí umožnit přiřazení rolí s různými přístupovými právy. Na základě přiřazené role bude možné uživatele odstínit od některých funkcí redakčního systému. Například uživatel v roli administrátora bude moci provádět operace se stránkami webu, zatímco uživatel v roli editora bude moci provádět operace s články apod.
- Systém bude navržen tak, aby zprostředkoval operace s webovými službami různých poskytovatelů. Zejména s webovými službami, které nabízí společnosti Facebook, Google a Twitter.

2.2 Nefunkční požadavky

Vlastnosti redakčního systému, které nepředstavují funkcionalitu:

- Instance systému na sobě musí být nezávislé.
- Aby bylo možné co nejlépe zohlednit požadavky koncového uživatele, musí být systém (respektive každá jeho instance) konfigurovatelný.
- Položky zmíněné ve funkčních požadavcích musí být reprezentované jako logické části aplikace (pluginy, moduly či komponenty), které bude možné používat nezávisle na sobě v různých instancích systému. Jestliže se tyto logické části budou objevovat ve více instancích systému, změna v jedné logické části se musí projevit ve všech instancích.
- Systém musí být navržen takovým způsobem, aby mohl implementátor doplnit jakoukoliv funkcionalitu, kterou koncový uživatel vyžaduje. Zpravidla ve formě zmíněné logické části aplikace.
- Z hlediska nasazování i údržby systému je důležité, aby systém umožňoval co nejméně komplikovanou implementaci požadovaných logických částí aplikace.
- Systém musí být navržen tak, aby bylo možné změnit grafický design stránek bez nutnosti zásahu do logiky aplikace.
- Doba potřebná pro zaškolení koncového uživatele systému na využívání veškeré funkcionality, kterou jeho instance systému poskytuje, by měla být řádově v desítkách minut.
- Systém musí být navržen tak, aby umožnil plánované spouštění operací.
- Implementátor musí mít přístup k veškerému zdrojovému kódu redakčního systému.

2.3 Požadavky z pohledu koncového uživatele

Následující body shrnují hlavní požadavky z hlediska uživatele, na kterého je vytvářený redakční systém cílený:

- Práce v systému musí vyžadovat nízké nároky na počítačovou gramotnost. Koncový uživatel by měl být schopen spravovat svůj web po krátkém zaškolení.
- Provádění akcí, které bude uživatel v dané instanci systému požadovat, musí být co nejvíce přímočaré. Pokud bude uživatel chtít upravovat obsah stránky, mělo by to být možné přímo na dané stránce. To znamená bez zbytečného přecházení do jiné části webu.

- Každá instance systému musí obsahovat jen uživatelem vyžádané funkce. Větší než žádaný rozsah funkcionality by totiž z pohledu uživatele představoval zbytečnou kognitivní zátěž.

K usnadnění správy webových stránek uživatele má přispět také integrace webových služeb. Jednotlivci či organizace mají kromě vlastních webových stránek často svůj účet také například na Twitteru, Facebooku nebo Googlu. Je tedy žádoucí, aby redakční systém využil potenciál, který webové služby poskytované těmito společnostmi nabízejí a umožnil také aktualizaci či synchronizaci obsahu těchto médií s obsahem webových stránek.

2.4 Požadavky z pohledu implementátora

Z nefunkčních požadavků je patrné (viz podkapitola 2.2), že je nutné při vývoji systému zohlednit také potřeby implementátora.

Jedním z nejdůležitějších požadavků je snadná přizpůsobitelnost systému potřebám koncového uživatele. Úkolem implementátora je tyto potřeby zohlednit. To bude v mnoha případech spojeno nejen s konfigurací instance systému a změnou šablon, ale také s úpravou nebo tvorbou nových částí systému. Systém tedy musí být navržen tak, aby provádění těchto úkonů představovalo co nejmenší komplikaci a implementátor mohl rychle nasazovat nové instance systému.

Veškeré pokročilejší úpravy webu (zásah do struktury webu, změna vzhledu apod.) by měl mít v kompetenci pouze implementátor webu. To proto, že některé změny tohoto typu mohou mít negativní dopad, který by uživatel systému nemusel neočekávat.

Redakční systém musí také poskytovat rozhraní v podobě REST API, které umožní propojení externích služeb. Toto rozhraní musí být navrženo takovým způsobem, aby bylo možné jej snadno rozšířit o operace s položkami, které bude koncový uživatel v systému vyžadovat.

3 Analýza běžně používaných redakčních systému

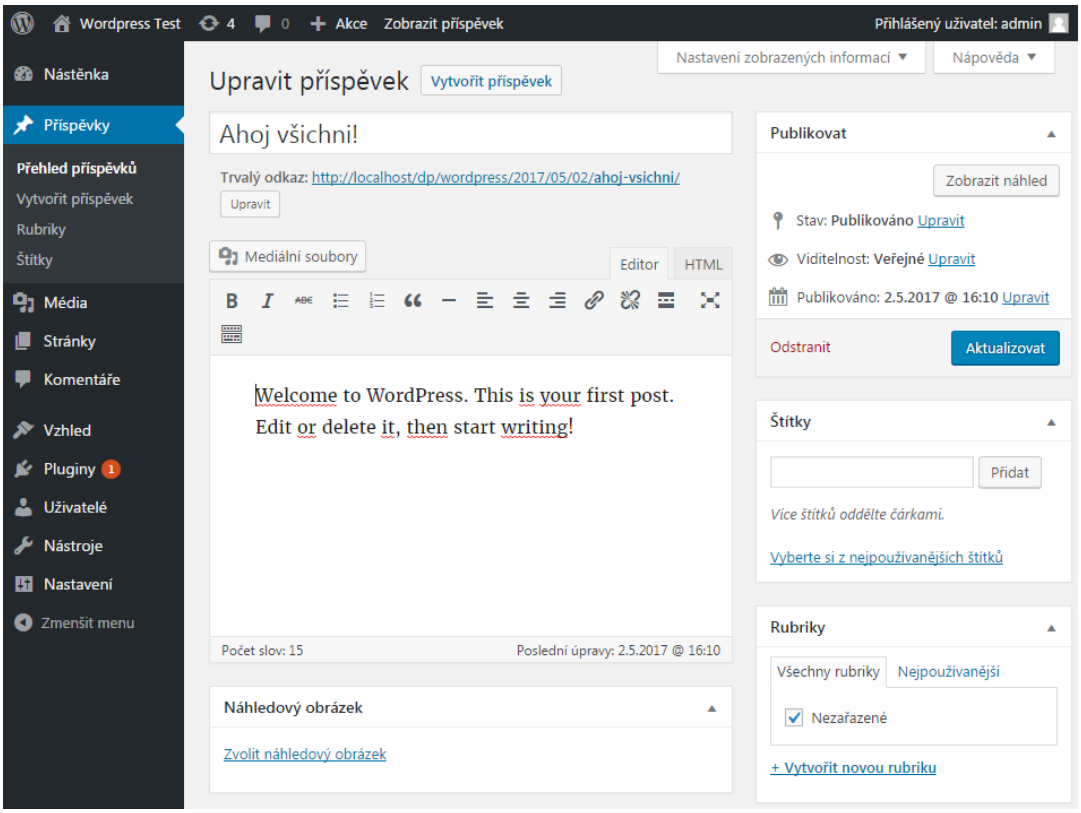
Jako možné řešení se nabízí využít běžně používané redakční systémy. Cílem této kapitoly je tedy zjistit, jestli existuje takový systém, který by splňoval požadavky specifikované v kapitole 2.

V následujících podkapitolách jsou diskutovány tři celosvětově používané redakční systémy. Jde o trojici systémů, které mají podle aktuálních statistik největší podíl na trhu (Q-Success, 2017). Všechny tři jsou dostupné zdarma.

3.1 WordPress

Jde o nejpoužívanější redakční systém. Podle statistik, které na svých stránkách uvádí Q-Success (2017), je tento systém nasazen na 58,9 procentech webů, které používají redakční systém, což je 27,9 % všech webových stránek.

Svoje využití našel tento systém také u projektů významných společnosti jako *CNN*, *Facebook*, *Microsoft*, *Spotify* nebo *TED* (WordPress, 2017a). Nutné je však podotknout, že některé tyto společnosti využívají placenou službu *WordPress.com VIP*, kde stojí nejlevnější program v přepočtu přibližně 125 000 Kč měsíčně (WordPress, 2017b).



Obrázek 1: WordPress – administrační rozhraní.

Aktuální verze (psáno 8. května 2017) tohoto systému je 4.7.4. K aktualizaci dochází v průměru každých 42 dní (Combell, 2017).

WordPress (2017c) na svých stránkách uvádí, že začal jako systém určený pro blogy a v tomto ohledu má také stále prvenství (BuiltWith, 2017). Čistá instalace tohoto systému (ze které pochází obrázek 1) nabízí poměrně omezené možnosti, které odpovídají potřebám uživatele, který vyžaduje právě blog nebo velmi jednoduchou webovou stránku. Po zavedení vhodných pluginů je však možné využít tuto aplikaci jako komplexní redakční systém (WordPress, 2017d).

Kromě snadné škálovatelnosti s pomocí pluginů jmenuje Wordpress (2017d) jako klíčové vlastnosti především jednoduchost správy obsahu webu, možnost správy uživatelů, pokročilou SEO optimalizaci, vysokou míru zabezpečení, responzivní design stránek, podporu více než 70 jazyků a velké množství přispěvatelů.

O velké popularitě WordPressu svědčí také početné množství pluginů a šablon, které byly pro tento systém vytvořeny. Přímo na stránkách WordPressu je zdarma k dispozici přibližně 50 tisíc pluginů a přes 4,6 tisíc šablon. Na Internetu je ale možné najít i velké množství placených šablon. Například na webu `themeforest.net` je ke stažení více než 9,6 tisíc šablon (Envato, 2017a).

Mezi počítanými jsou však také pluginy a šablony, které již nebyly delší dobu aktualizovány a mohou tedy být kompatibilní pouze se staršími verzemi systému. WordPress ale ve vyhledávání neumožňuje odfiltrovat neaktuální verze, takže lze jen těžko zjistit jejich počet.

3.2 Joomla

Se 7 procenty je Joomla druhým nejpoužívanějším redakčním systémem (Q-Success, 2017). Ačkoliv je tento propad ve srovnání s WordPressem výrazný, používají i tento systém významné organizace, mezi které patří například *Harvardova univerzita*, *IKEA*, *Holiday Inn*, *Linux.com* nebo kongresové noviny *The Hill* (Jetimpex, 2017).

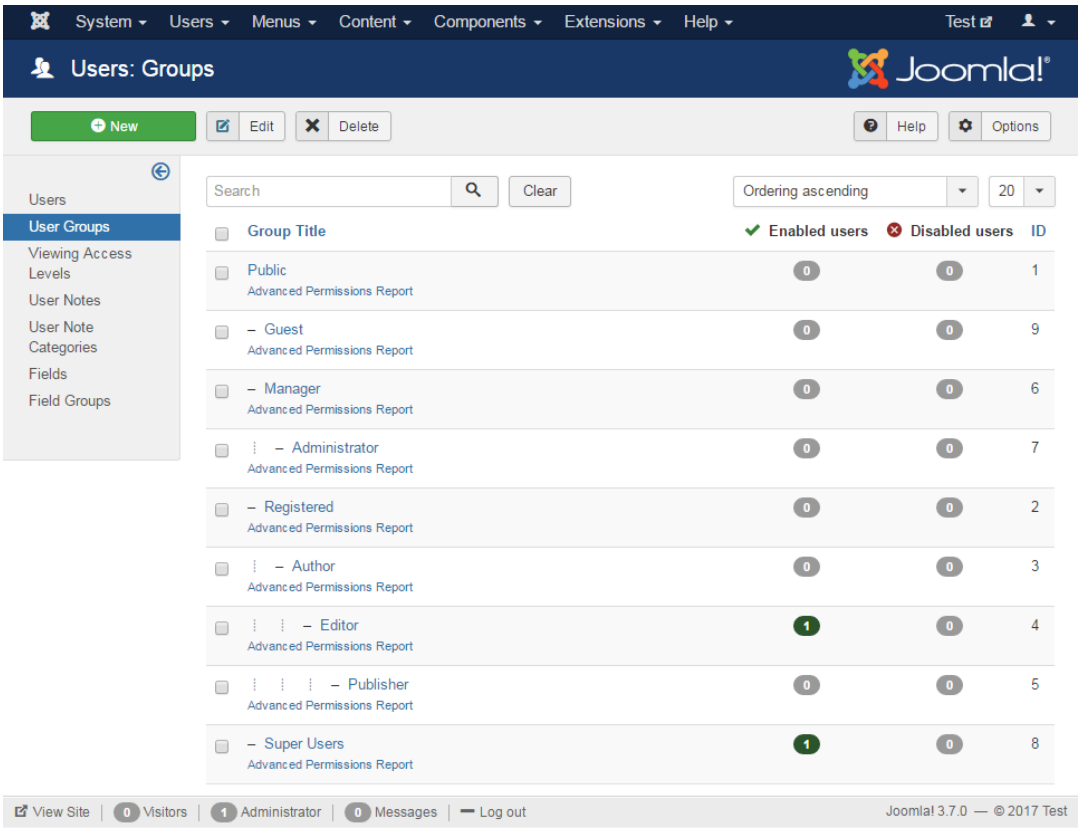
Aktuální verze (psáno 8. května 2017) tohoto systému je 3.7.0. K aktualizaci dochází v průměru každých 51 dní (Combell, 2017).

Stejně jako WordPress i Joomla uvádí jako svoje hlavní přednosti jednoduchost použití a rozšiřitelnost (Open Source Matters, 2017a). Možnosti tohoto systému jsou však v základu (tzn. i bez pluginů) rozsáhlejší než ve WordPressu. Administrační rozhraní je tedy o to složitější.

Jako další důležité vlastnosti uvádí autoři systému například podporu více než 65 jazyků, rozsáhlé možnosti při vytváření struktury webu (správa menu, kategorií či štítkování), dobrou uživatelskou podporu, pokročilé prohledávání obsahu webu a rozsáhlý Access Control List. Ukázka administračního rozhraní je zachycena na obrázku 2 (Open Source Matters, 2017b).

Joomla má z vybraných tří redakčních systému nejnižší počet pluginů – necelých 8 tisíc. Ani zde není možné jednoduše zjistit kolik je jich aktuálních. V rámci stránky s pluginy se lze pouze dozvědět, že jsou všechny kompatibilní s verzí 3.0. Nutné je však podotknout, že od vydání této verze uplynulo pět let (Open Sour-

ce Matters, 2017c). Při výběru pluginu je tedy nutné sledovat především datum poslední aktualizace.



Obrázek 2: Joomla – administrační rozhraní.

Tento systém na svých stránkách nenabízí ke stažení žádné šablony. Na Internetu je opět možné vyhledat šablony i pro tento systém, ale těžko se dá odhadnout jejich počet. Na zmiňovaném webu `themeforest.net` jich je k dispozici necelý 1 tisíc (Envato, 2017b). Joomla tedy také v tomto ohledu ve srovnání s WordPressem strádá.

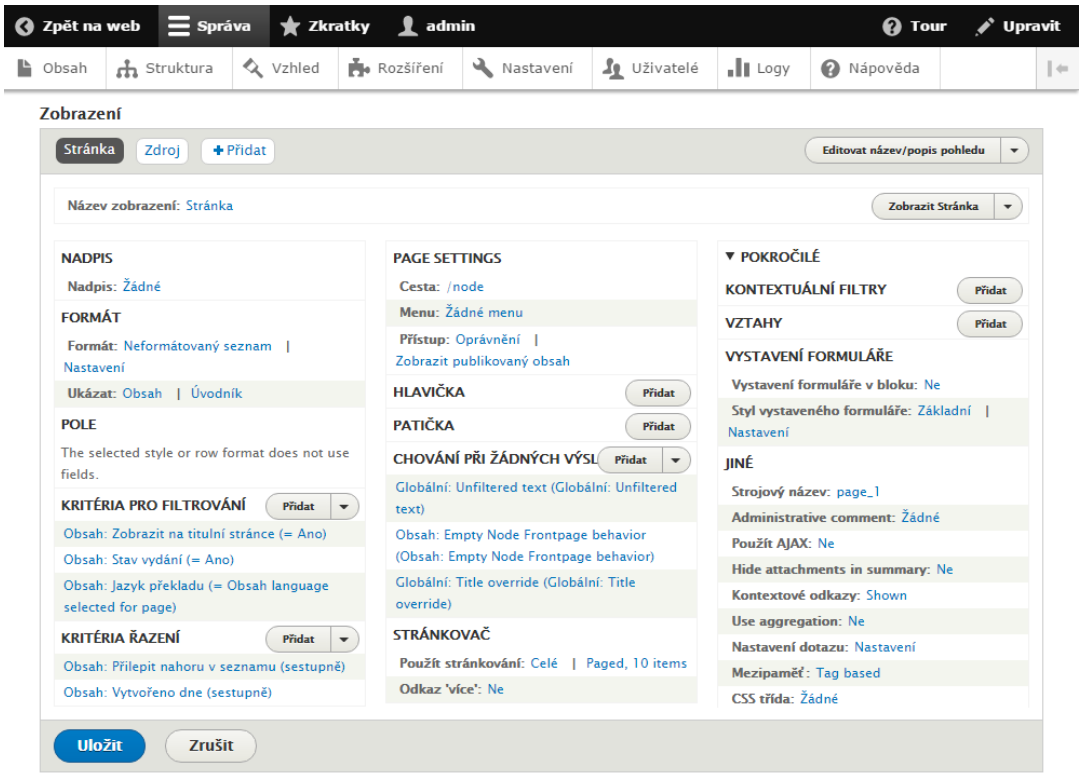
3.3 Drupal

Tento redakční systém má 4,7% podíl na trhu a je posledním systémem, který je nad hranicí 2,5 procenta (Q-Success, 2017). Drupal využívají celosvětově známé společnosti jako je například *Twitter*, *Tesla*, *Cisco*, *BBC* a řada amerických univerzit či vládních webů včetně webových stránek *Bílého domu* (Buytaert, 2017a).

Aktuální verze (psáno 8. května 2017) Drupalu je 8.3.2. K aktualizaci dochází v průměru každých 36 dní (Combell, 2017).

Také v Drupalu je kladen důraz na flexibilitu a škálovatelnost systému. Rozsah nastavení je zde však ještě větší než u Joomly. Na obrázku 3 je například k vidění

nastavení zobrazení stránky. Některé další vlastnosti jsou srovnatelné s předcházejícími systémy. Jde například o podporu 94 jazyků, dobré zabezpečení, responzivní design stránek a snadné vytváření obsahu (Buytaert, 2017b). Disponuje ovšem také zabudovaným RESP API, usnadňuje napojení na platební brány či CRM a ERP řešení (Buytaert, 2017c).



Obrázek 3: Drupal – administrační rozhraní.

Na stránkách Drupalu je možné najít přes 37 tisíc pluginů a 1500 šablon. Drupal je jediný systém, na jehož webu je možné odfiltrovat neaktuální pluginy a šablony. Ve výpisu pak zůstane přibližně 3,2 tisíc pluginů a 200 šablon (Buytaert, 2017d).

3.4 Propojení s webovými službami

Ve všech uvedených redakčních systémech je možné v určitém rozsahu integrovat webové služby z pomocí pluginů. V této podkapitole jsou tedy uvedeny nejzajímavější pluginy z hlediska funkcionality.

Vybrány byly pouze pluginy, jejichž poslední aktualizace proběhla maximálně půl roku zpátky a mají nadprůměrné hodnocení. V případě, že byly nalezeny pluginy, které mají srovnatelnou funkcionalitu, byl vybrán ten, u něhož bylo zaznamenáno lepší hodnocení, více stažení nebo nižší cena. Vyloučeny byly triviální pluginy, které pouze obsahují jiné snadno implementovatelné pluginy jako například *Page Plugin* od Facebooku (Facebook, 2017a).

3.4.1 Výpis příspěvků ze sociálních sítí

Výpis příspěvků ze sociálních sítí je uskutečnitelný s pomocí pluginů u všech tří analyzovaných redakčních systémů. Ve WordPressu (2017e; 2017f; 2017g) lze k tomuto účelu nainstalovat plugin *Custom Facebook Feed* (pro výpis Facebookových statusů), *Custom Twitter Feeds* (pro výpis tweetů) a *The Instagram Feed* (výpis uživatelových fotek).

Joomla umožňuje pouze výpis tweetů s využitím placeného pluginu *Latest Tweet*, který vyžaduje platbu 725 Kč za každou webovou stránku, kde bude nasazen (WebKul, 2017). Oproti zmíněným WordPress pluginům navíc dokáže zobrazit výpis příspěvků z více zadaných Twitter účtů zároveň.

Pro Drupal existuje plugin *Social Timeline*, který zobrazuje výpis příspěvků, statusů, videí či obrázků ze sociálních sítí, jako jsou například Twitter, Facebook, YouTube, Dribbble, Pinterest a Instagram (Buytaert, 2017e).

Ačkoliv všechny výše uvedené pluginy poskytují tu samou informaci, zásadně se liší v možnostech nastavení vzhledu. Tato vlastnost je poměrně důležitá, protože výpis příspěvků, který nebude zapadat do designu dané stránky, může zákazníka odradit od jeho používání. V tomto ohledu je nejvýhodnější zvolit uvedené pluginy pro WordPress, které nabízí nejrozsáhlejší možnost customizace.

Hlavní nevýhodou všech těchto pluginů ovšem je, že požadovaná data nestahují do systému a při každém načítání stránky je tedy stahují z API znova. To poměrně výrazně zpomaluje načtení žádaného obsahu. Tyto pluginy navíc nesplňují požadavek na možnost synchronizace webu s účty na sociálních sítích (viz podkapitola 2.3), protože zde není možné odesílat příspěvky z webu na tyto účty. Za tímto účelem je nutné instalovat další plugin.

3.4.2 Vkládání příspěvků do sociálních sítí

Tuto funkcionalitu nabízí plugin *NextScripts: Social Networks Auto-Poster* pro WordPress (Wordpress, 2017h). Funguje tak, že zároveň se vkládáním příspěvku v rámci redakčního systému vloží status na zvolené sociální sítě.

Nevýhodou je komplikovanost použití tohoto pluginu. Uživatele totiž není možné odstínit od složitějších nastavení, které se s tímto pluginem vážou. Ideální by byla možnost, kdy uživatel může před publikováním příspěvku pouze zaškrtnout, jestli chce vložit informaci o příspěvku také na sociální sítě. Místo toho je zde panel s větším množstvím nabídek, které by uživatele mohly obtěžovat.

Další nevýhodou je, že se informace o vložení statusu na sociální síť nikde neukládá. Po úpravě nebo odstranění příspěvku na webu tedy zůstane status beze změny.

3.4.3 Kalendáře a události

Zajímavou variantu kalendáře z hlediska propojení s webovými službami představuje plugin WordPressu *Simple Calendar – Google Calendar Plugin* (Wordpress,

2017i). Ten zobrazuje události, které jsou vytvořeny v rámci služby *Google Calendar*. Jeho největší předností (ve srovnání s iframe verzí kalendáře přímo od Googlu) je pokročilejší možnost úpravy vzhledu. Nedostatkem tohoto pluginu naopak je, že neumožňuje události do tohoto kalendáře vložit.

Z hlediska událostí pak stojí za zmínku také plugin *Import Facebook Events* (rovněž pro WordPress). S jeho pomocí je možné stahovat události z Facebooku do databáze systému. Pří zakoupení verze *Pro* lze provádět tuto operaci pro zadanou Facebookovou stránku automatizovaně v určitém časovém intervalu. Jeho cena se pohybuje v rozmezí od 1 225 Kč (pro jeden web) do 3 725 Kč (pro 20 webů) pro první rok používání a v následujících letech za 50 % této ceny (Xylus Themes, 2017).

Importované události lze zobrazit v některém z kalendářových pluginů WordPressu (Xylus Themes, 2017). Těmi jsou například *The Events Calendar*, *Events manager*, *All-in-One Event Calendar* apod.

3.4.4 Fotoalba

Zobrazení fotoalb s využitím webových služeb je možné u WordPressu a Joomly. Nejvhodnější plugin, který tuto funkcionalitu umožní u Joomly se nazývá *JUX Social Gallery* (JoomlaUX, 2016). Zobrazuje fotoalba, která jsou umístěna na zadaném účtu Facebooku, Instagramu, Google+ a Flickru. Bohužel nelze zobrazit fotoalba všech zadaných účtů zároveň a možnosti customizace jsou ve srovnání s pluginy WordPressu opět omezenější. Tento plugin je k dispozici od částky 250 Kč (pro jednu doménu na dva měsíce) do 1 870 Kč (pro sedm domén na rok).

WordPress nabízí širší výběr. Nejpopulárnější jsou tyto pluginy:

- Photonic Gallery for Flickr, Picasa, SmugMug, 500px, Zenfolio and Instagram – podobně jako *JUX Social Gallery* umožňuje získat obrázky z více různých zdrojů (Wordpress, 2017j).
- Instagram Feed WD - Instagram Gallery for WordPress – plugin specializovaný pro obrázky z Instagramu. Kromě samotných obrázků zobrazuje také počet jejich kladných hodnocení a komentářů (Wordpress, 2017g).
- Srizon Facebook Album – umožňuje zobrazit galerii obrázků z Facebooku. Neplacená verze je však značně omezená. Zobrazuje pouze 25 obrázků pro každé album a nenabízí možnost odfiltrování uživatelem zadaných alb (Srizon, 2017). Za tyto vlastnosti je nutné zaplatit v přepočtu přibližně 500 Kč (jednorázově).

Výhodou prvního zmíněného pluginu je konzistence způsobu zobrazení a ovládacích prvků galerie pro všechny zdroje. Pokud by však uživatel chtěl na stránkách i alba z facebooku, musel by navíc použít plugin *Srizon Facebook Album*. Stejně jako *JUX Social Gallery*, ani *Photonic Gallery* nedokáže zobrazit obrázky z různých zdrojů v jedné galerii.

Další nevýhody jsou podobné jako u výpisu příspěvků ze sociálních sítí. S každým načtením stránky jsou obrázky načítány znova z API jejich poskytovatelů, což

může vést k jejich pomalejšímu načítání. Ani zde také není možné využít toto API k nahrání obrázků na zadaný zdroj či více zdrojů.

3.5 Shrnutí

Pokud bych měl zhodnotit vlastnosti těchto systémů bez nainstalovaných pluginů, nejkomplexnějším (ale nejméně přehledným) systémem by byl Drupal a nejsnáze ovladatelným (avšak nejméně komplexním) WordPress. Jak ale bylo zmíněno v podkapitole 3.1, pro WordPress existuje velké množství pluginů, které dělají i z tohoto systému komplexní nástroj s řadou užitečných možností. A pluginy jsou nepopiratelnou předností tohoto systému. Díky nim také WordPress splňuje největší měrou požadavky na integraci webových služeb (viz kapitola 2). S instalací pluginů se ovšem výrazně zhoršuje přehlednost a přímočarost ovládání také u tohoto systému.

Velký rozsah možností nastavení těchto systémů (potažmo jejich pluginů) má své klady i zápory. Jestliže jsou tyto možnosti dostatečně rozsáhlé pro splnění reálných požadavků koncových uživatelů, může je při své práci s výhodou využít implementátor systému. Zároveň by však bylo potřeba od těchto možností odstínit koncového uživatele, protože nastavování systému má mít v kompetenci pouze implementátor. Administrační část webu, kde lze veškerá nastavení provádět, ovšem poskytuje také možnost správy obsahu webu, kterou vyžaduje koncový uživatel. Do této části webu tedy musí uživatel přejít, a to i kvůli provedení tak triviální akce, jako je například odstranění příspěvku. Tato vlastnost systému odporuje požadavkům z pohledu koncového uživatele specifikovaným v podkapitole 2.3.

Ačkoliv zmíněné pluginy WordPressu nabízí poměrně zajímavé možnosti v oblasti integrace webových služeb, stále jsou zde určitá omezení. Většina uvedených pluginů (viz podkapitola 3.4) poskytuje pouze možnost získávání dat z daných API. V některých případech však může koncový uživatel vyžadovat vkládání, úpravu či odstraňování těchto dat. Z pohledu implementátora je zase například nepraktické zadávat ID a secret řetězce Facebookové aplikace do několika konfigurací pluginů, když by mohly být v jednom konfiguračním souboru a odtud podle potřeby získávány.

I tak ovšem uvedené pluginy představují inspiraci pro návrh vlastního řešení. Například funkcionalita pluginu *Import Facebook Events* (viz podkapitola 3.4.3) by mohla být rozšířena o možnost importovat událost také do Google kalendáře. Tento způsob provázání více webových služeb mezi sebou je obecně funkcionalitou, kterou nenabízí žádný z analyzovaných pluginů, přitom však vede k usnadnění práce koncového uživatele.

Pro implementátora je také nevýhodou, že nemá kontrolu nad vývojem těchto systémů a pluginů. Mohl by se sice na jejich vývoji podílet, požadované změny by však nemusel prosadit a i kdyby se mu to podařilo, mohlo by trvat nepřijatelně dlouho, než se objeví aktualizace, která bude tyto změny zahrnovat.

4 Návrh řešení

Ze souhrnu provedené analýzy je patrné, že dostupné redakční systémy k dosažení stanoveného cíle využít nelze. Je proto nutné realizovat vlastní řešení.

Účelem této kapitoly je volba co nejvhodnějších technologií a návrh takových postupů, ze kterých bude možné vycházet při implementaci požadované aplikace.

4.1 Zvolené technologie

Pro realizaci redakčního systému byly zvoleny technologie, které jsou typické pro webové aplikace. Pro vytvoření frontendu tedy jazyky HTML5, CSS3 a JavaScript (a vhodné knihovny) a na straně backendu databázový systém MySQL a PHP framework Nette[1].

4.2 Struktura aplikace

4.2.1 Komponenta

Komponenta je základním stavebním prvkem webové aplikace tvořené ve frameworku Nette. Komponentou je každá třída, která dědí od třídy `Component` (ta je umístěna ve jmenném prostoru `Nette\ComponentModel`). Které třídy tedy lze pokládat za komponenty znázorňuje obrázek 4.

```
Nette\ComponentModel\Component  { IComponent }
|
+- Nette\ComponentModel\Container  { IContainer }
   |
   +- Nette\Application\UI\Component  { ISignalReceiver, IStatePersistent }
      |
      +- Nette\Application\UI\Control  { IPartiallyRenderable }
         |
         +- Nette\Application\UI\Presenter  { IPresenter }
```

Obrázek 4: Hierarchie dědičnosti komponenty.
Zdroj: Nette Foundation, 2017a.

Zmíněná třída `Component` (která je společným předkem všech tříd v uvedené hierarchii) je třídou abstraktní, jejíž implementace tvoří základ z hlediska provázání vytvořených komponent s jejich rodiči. Metody, které třída `Component` poskytuje, dále využívá třída `Container`. Ta rozšiřuje svého předchůdce o operace pro přidávání, odebírání a získávání komponent. Důležitá je skutečnost, že `Container` může obsahovat libovolné množství (různých) komponent. Díky tomu mohou komponenty tvořit stromovou strukturu, kde může být jedna komponenta tvořena potřebným

[1]Konkrétně byla zvolena verze Nette 2.4, která vyžaduje PHP 5.6 nebo vyšší a je rovněž kompatibilní s PHP 7.1.

množstvím dalších. Jako příklad využití lze uvést formulář, který je složen z jednotlivých formulářových prvků.

`Container` je tedy základem pro realizaci komplexních komponent, stále ovšem nejde o třídu, která by byla schopná odpovídat na požadavky uživatele. Možnost obsluhy těchto signálů[2] tak zvanými *handler* metodami doplňuje třída `Component` (tentokrát ve jmenném prostoru `Nette\Application\UI`). Děděním z této třídy sice je možné vytvořit komponentu, která umožňuje interakci s uživatelem (což je také důvodu umístění této třídy ve jmenném prostoru `UI`), i tak je ale její využití značně omezeno, protože jde o komponentu nevykreslitelnou. To znamená, že nedisponuje metodami, které by umožnily vykreslování (respektive překreslování při AJAX požadavku) přiřazené šablony.

Prvním ze dvou zástupců komponent, které tyto operace umožňují, je třída `Control`. Ta představuje logickou část stránky, kterou je možné podle potřeby opětovně použít na různých místech webové aplikace. Znovupoužitelnost a možnost vytváření stromové struktury jsou klíčové vlastnosti komponent, které výrazně usnadňují návrh, tvorbu a údržbu aplikace. Při tvorbě netriviální webové aplikace (kterou redakční systém bezesporu je), je sestavení aplikace pomocí těchto prvků de facto nezbytné. V následujícím textu se pod pojmem *komponenta* rozumí potomek třídy `Control` (ne `Component`). Zvláštním případem vykreslitelné komponenty je presenter.

4.2.2 Presenter

Jelikož třída `Presenter` rozšiřuje `Control`, nabízí se také varianta sestavit celou aplikaci z presenterů. Vývoj redakčního systému by se tak ale výrazně zkomplikoval. To proto, že presentery nejsou navrženy pro znovupoužívání a ani k přidávání do ostatních komponent. Nebylo by tedy o možné dekomponovat web na menší samostatně použitelné celky. Údržba kódu, který realizuje část stránky společnou pro více webů, by pak neznamenala zásah do jedné oddělené komponenty, ale určité části presenteru, která by se navíc mohla web od webu lišit.

Přesto má však presenter výhodné vlastnosti, díky kterým najde v této aplikaci využití. Je totiž vrstvou aplikace, jejímž účelem je převzít požadavek přeložený routerem[3] z HTTP požadavku a vygenerovat odpověď. V podkapitole 4.2.1 bylo uvedeno, že komponenty umístěné ve jmenném prostoru `Nette\Application\UI` mohou odpovídat na tzv. signály. Presenter představuje tu část aplikace, která zachycuje a předává tyto požadavky komponentám, které je mají zpracovat – tedy v případě, že jsou v něm obsaženy.

[2]Signál (nebo také subrequest) je komunikace se serverem v rámci jednoho view – to znamená, že nemění view. Může jej přijímat komponenta, presenter nebo jakýkoliv objekt, který implementuje rozhraní `ISignalReceiver` (Nette Foundation, 2017a.).

[3]Router je část webové aplikace, která realizuje obousměrné překládání mezi URL a aplikačním požadavkem (Nette Foundation, 2017b.).

Proto lze s výhodou využít vlastností, které presenter dědí z třídy `Container` a vložit do něj potřebné komponenty. Presenter se tak stane tzv. *komponentovým kontejnerem* a bude tvořit kořen stromové struktury těchto komponent.

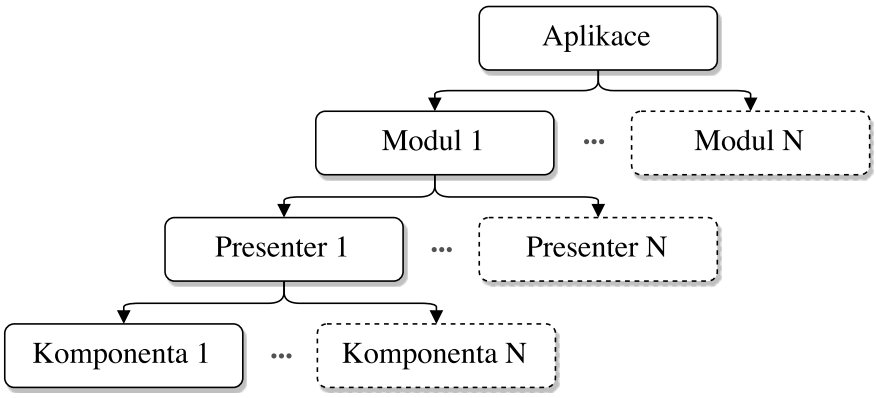
Unikátní vlastností presenteru také je, že jako jediným typ komponenty může měnit view. Nejde sice o nutnost, protože o změnu vzhledu stránky se mohou postarat jednotlivé komponenty, použití více view ovšem může usnadnit práci například v případě, kdy je potřeba pro všechny komponenty v presenteru předepsat určité chování (může jít o změnu vzhledu, zpřístupnění nebo zakázání některých akcí atp.).

Nette Foundation (2017c) uvádí, že *presenter reprezentuje instanci webové stránky*. Toto tvrzení může svádět k myšlence, že pro každou stránku webu musí být vytvořen zvláštní presenter. Cílem práce je však realizovat redakční systém, který umožní vytvářet stránky dynamicky. Vhodnější výklad proto je, že je presenter článkem webové aplikace, který se stará o to, aby byl zobrazen požadovaný obsah stránky. Co bude tímto obsahem, může být podmíněno a jeden presenter tak může zobrazovat potřebné množství webových stránek. Počet presenterů bude tedy stejný pro všechny weby (viz podkapitola 4.2.4), na kterých bude systém nasazen. Díky tomu se může implementátor soustředit především na tvorbu komponent, které budou tvořit požadovaný obsah stránek.

4.2.3 Modul

Dalším prvkem, kterým Nette umožňuje definovat strukturu aplikace, je *modul*. Každý modul může obsahovat jiné presentery, komponenty, šablony a modely. Použití modulů v kombinaci s vhodným routováním může nabídnout různé způsoby komunikace s aplikací. Zatímco jeden modul bude představovat web se standardním webovým rozhraním, jiný modul dovolí provádět akce pouze přes příkazovou řádku.

Podobně jako komponenty, i moduly do sebe mohou být zanořovány a tvořit tak submoduly. U této aplikace ale postačí rozdělení do modulů. Jak vypadá hierarchie doposud diskutovaných prvků aplikace při použití modulů, naznačuje obrázek 5.



Obrázek 5: Hierarchie prvků modul, presenter a komponenta.

4.2.4 Navržená struktura

Následující body popisují členění do modulů a presenterů:

- `FrontModul` – nejobsáhlejší modul aplikace. Jde o část aplikace s běžným webovým rozhraním. Sem budou patřit všechny komponenty a šablony. Tvoří jej tyto tři presentery:
  - `BasePresenter` – abstraktní třída, která je společným předkem pro další dva uvedené presentery. Bude tedy obsahovat funkcionalitu, která je potřebná napříč celým tímto modulem. To znamená například akce umožňující přihlašování a odhlašování nebo některé komponenty určené k administraci webu.
  - `PagePresenter` – tento presenter je navržen k reprezentaci těch stránek, které představují jednotlivé sekce webu. Může obsahovat výpisy různých položek (článků, fotoalb, událostí atp.).
  - `ItemPresenter` – presenter vykreslující položky, které lze otevřít na samostatné stránce.

  Příkladem sekce webu může být tedy stránka s výpisem článků. Tu realizuje `PagePresenter`. Pokud uživatel otevře některý z těchto článků, stránka, na které se ocitne, je zprostředkována třídou `ItemPresenter`. Rozdíl mezi sekcemi a položkami zachycuje také obrázek v příloze A.
- `ApiModul` – tato část aplikace je určená k propojení externích služeb s využitím architektury REST. Této problematice se blíže věnuje podkapitola 5.3.
- `CliModul` – tento modul obsahuje jediný presenter, který poskytuje rozhraní v podobě akcí, které mají být volány Cronem. Aby nedocházelo k nevyžádanému volání těchto akcí, je dostupný pouze přes příkazovou řádku. Využití tohoto modulu souvisí s integrací webových služeb (viz podkapitola 4.4.7). Téma týkající se implementace tohoto modulu je rozvedeno v podkapitole 5.2.3.

`FrontModul` bude jako jediný využívat více view. Výchozí *default* view vykreslí stránku tak, jak ji vidí běžný návštěvník webu. Při použití *admin* view pak bude uživateli zpřístupněno administrační rozhraní (samozřejmě jen tehdy, pokud má jeho uživatelský účet příslušná privilegia).

Tato struktura modulů a presenterů bude neměnná pro všechny webové stránky, na kterých bude navrhovaný redakční systém nasazen. Poslední článek hierarchie aplikace, kterým jsou komponenty modulu `FrontModul`, se však bude lišit podle potřeb koncového uživatele systému, protože právě použité komponenty budou určovat podobu a funkcionalitu této aplikace.

4.3 Struktura databáze

Souhrn předpokladů, na kterých je založen návrh databáze:

- Jedna z tabulek bude obsahovat záznamy týkající se jednotlivých sekcí webu.
- Sekce webu budou mít jistě různý obsah. Zároveň však může být více sekcí tvořeno stejnými komponentami. V tom případě lze říci, že jde o sekce stejného typu. Příkladem mohou být sekce s články. Články sice budou různé, ale vykreslovat je bude stejná komponenta. Protože se tedy typ sekce může opakovat, musí být v databázi také číselníková tabulka s těmito typy.
- Dále bude potřeba vytvářet tabulky, ve kterých bude uchováván obsah položek.
- U některých položek bude možné povolit vkládání komentářů. To vyžaduje příslušnou tabulku s vazbou na tyto položky.
- Dále je žádoucí, aby bylo možné na stránkách vytvářet galerie obrázků, z čehož vyplývá potřeba další tabulky, do které bude možné vkládat záznamy o vložených obrázcích.
- Je také nutné vytvořit tabulku, která bude uchovávat údaje o uživatelích. Tato tabulka bude mít vazbu na položky, u nichž bude třeba dohledat, kdo je jejich autorem.
- Na výše uvedenou tabulku uživatelů bude mít vazbu číselníková tabulka uživatelských rolí.

Největší komplikaci představuje návrh tabulek, které mají uchovávat obsah položek. Tento obsah je sice u každé položky odlišný (proto je nutné jednotlivé položky podle typu rozdělit do různých tabulek), přesto je však potřeba, aby měli různé položky vazbu na stejné tabulky. Pro lepší představu uvedu příklad: Jak u položky *článek*, tak u položky *událost* mají být vkládány komentáře, což vyžaduje vazby obou položek na tabulku s komentáři.

Možné varianty řešení tohoto problému, včetně konečné podoby návrhu databáze, popisují následující podkapitoly.

4.3.1 První varianta

Tato verze je zachycena na obrázku 6. Jde o databázi jednoho z prvních webů, kde byl vyvíjený redakční systém nasazen v rámci testování. Rozbalené tabulky představují jednotlivé typy položek.

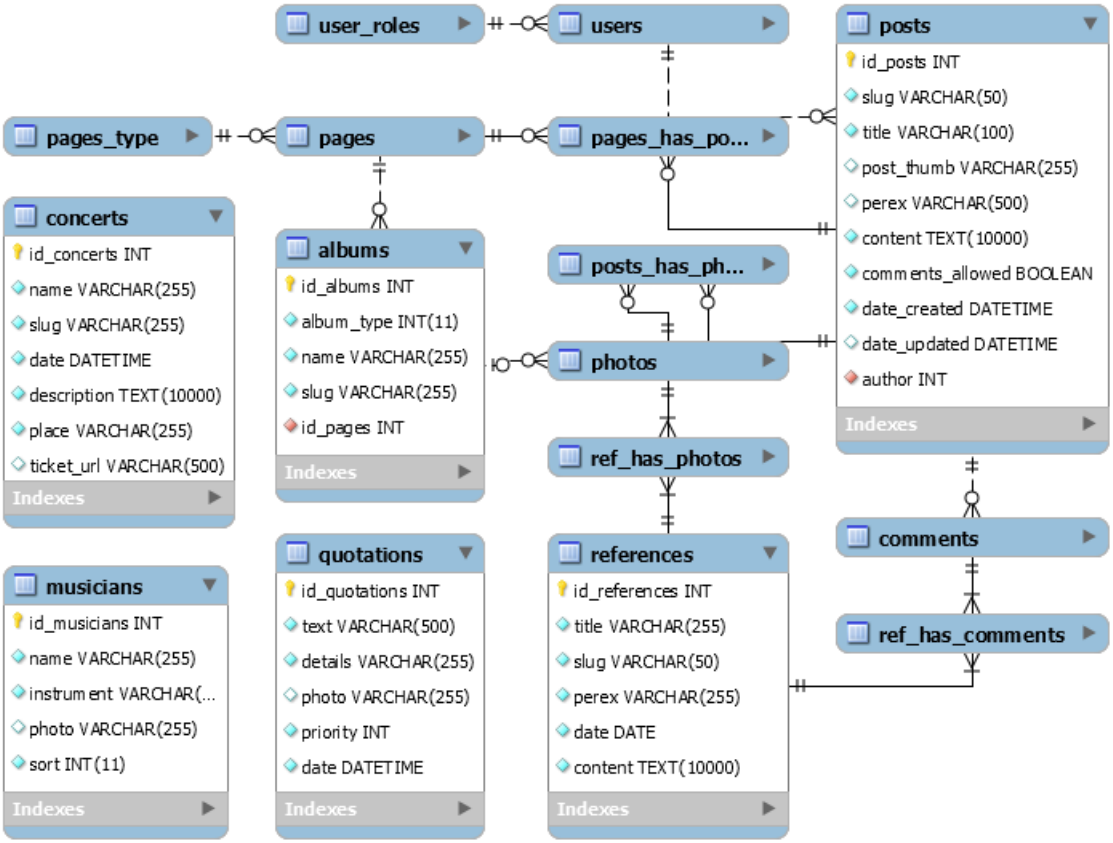
Ačkoliv byl tento způsob uchování dat provozuschopný, záhy se ukázal jako poměrně neefektivní. Dvě největší nevýhody tohoto řešení jsou tyto:

1. Jak znázorňuje obrázek 6, tabulka `references` obsahuje položky, které mohou obsahovat fotky a komentáře. Aby toho šlo dosáhnout, bylo zapotřebí zde vytvořit asociační tabulky `ref_has_photos` a `ref_has_comments`. Další možnost

by byla vytvořit v tabulce `photos` a `comments` atribut s cizím klíčem na tabulku `references`, nebo vytvořit asociační tabulku, která bude obsahovat cizí klíče všech tabulek, které mají mít referenci na tabulku `comments` apod. Ani jedna z těchto variant ovšem není vhodná, protože potenciálně (tzn. s každým dalším typem položky) vede k vytváření zbytečně velkého množství tabulek nebo atributů.

2. Tabulky s položkami mají často společné atributy. Těmi jsou například `slug`[4], `title` nebo `name` a `date`. Z toho vyplývá, že by tyto atributy mohly být dostupné v jedné tabulce, která bude společná pro všechny položky.

Pokud by si tedy uživatel systému vyžádal změnu, jako je třeba povolení komentářů u koncertů, bylo by nutné provést relativně velký zásah do databáze. Kvůli dosažení vyšší pružnosti systému bylo třeba podniknout určité úpravy.



Obrázek 6: První verze návrhu databáze.

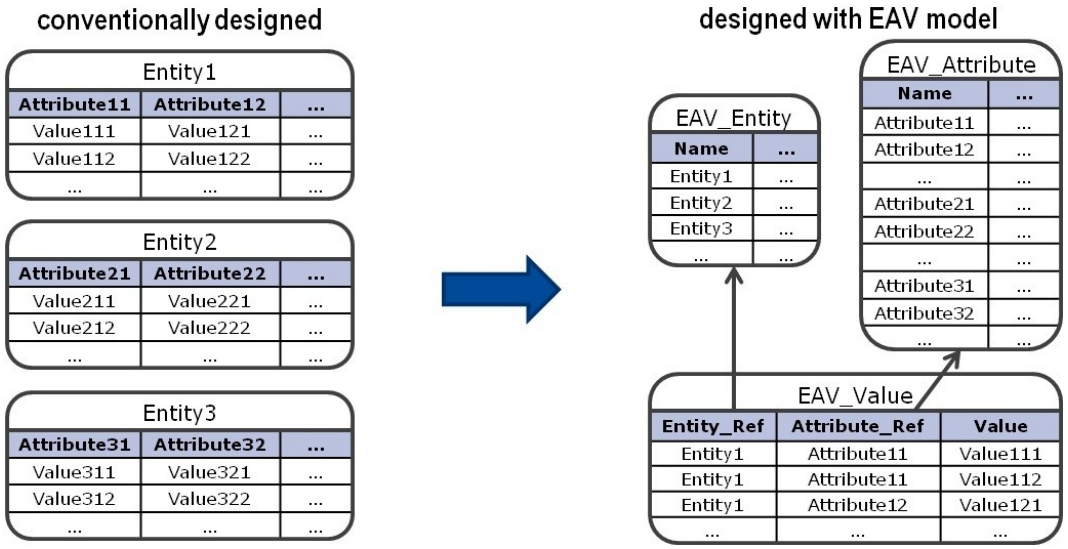
[4]Slug je část URL, která identifikuje webový zdroj s pomocí člověku srozumitelných klíčových slov (Wikipedia Foundation, 2017). Příklad: `http://www.domena.cz/toto-je-slug`

4.3.2 Entity-Attribute-Value model

Další zvažovanou variantou bylo využití tzv. Entity-Attribute-Value modelu (ve zkratce EAV modelu). Je to přístup, který umožňuje nahradit teoreticky neomezené množství tabulek s různými atributy třemi typy tabulek, z jejichž označení je složen název tohoto modelu (Foster a Godbole, 2016):

- **Entity** – obsahuje záznamy, které specifikují jednotlivé entity. Šlo by tedy o typy položek (článek, událost, album apod.).
- **Attribute** – definuje atributy entit.
- **Value** – propojuje obě výše zmíněné tabulky, přičemž vždy uchovává hodnotu pro kombinaci entita-atribut.

Obecnou podobu EAV modelu zachycuje obrázek 7. Takto by bylo možné dosáhnout sjednocení obsahu všech typů položek do jedné společné struktury. Vytvoření dalšího typu položky znamená vložení několika řádků do již existujících tabulek, namísto vytváření tabulky nové. Tabulka `EAV_Entity` může navíc obsahovat také atributy, které jsou pro všechny entity (položky) společné. Zde by tedy mohli být například atributy typu *slug*, *date*, *name* či *title*.



Obrázek 7: Příklad EAV modelu.
Zdroj: Löper a kol., 2013.

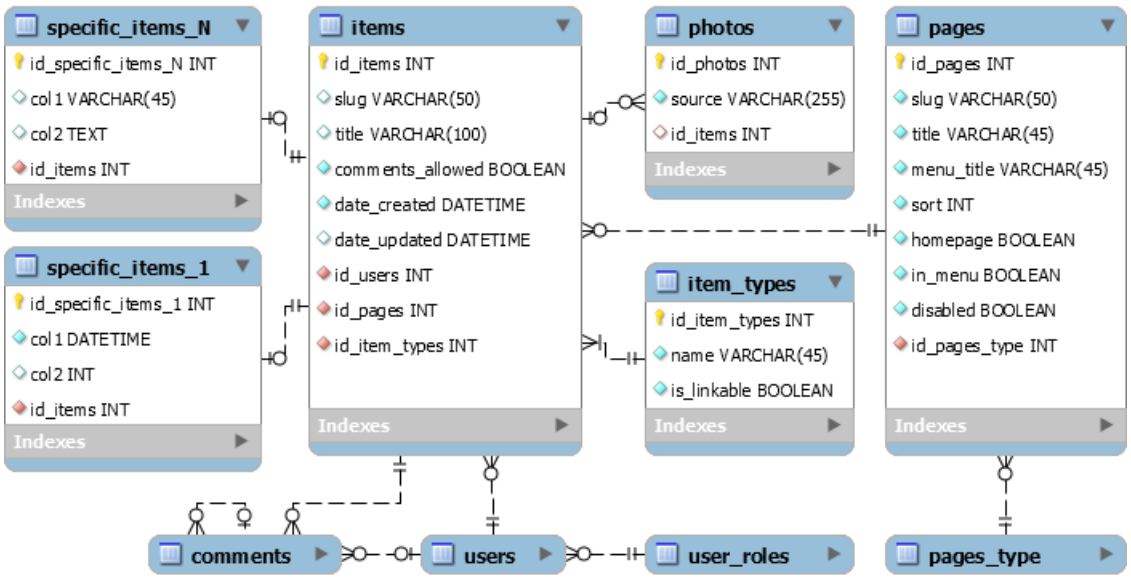
Ovšem i EAV model má své stinné stránky. Obrázek 7 totiž pouze nastiňuje základní myšlenku tohoto modelu. Při vlastním návrhu brzy vyvstane problém, že ve sloupci `value` tabulky `EAV_Value` je potřeba uchovávat hodnoty s různými datovými typy. Tento problém lze řešit více způsoby. Löper a kol. (2013) například navrhuje vytvořit zvláštní tabulky typu *value* pro každý používaný datový typ. Andreev (2012) zase doporučuje vytvořit vlastní sloupec pro všechny potřebné datové typy rovnou v tabulce typu *value* a hodnoty pak vkládat do příslušného sloupce.

Dále je také potřeba zohlednit, že každý atribut nemusí být povinný. Pak je vhodné do tabulky *attribute* přidat sloupec, který bude o (ne)povinnosti atributu informovat.

Se zavedením EAV modelu jsou tedy spjaté problémy, které sice je možné řešit, ovšem na úkor vyšší režie. Na aplikační vrstvu by tedy byly kladeny vyšší nároky ať už z hlediska vkládání, nebo při pozdější selekci hodnot. To je také hlavní argument proti použití této varianty.

4.3.3 Finální návrh

Jako nejvhodnější se nakonec ukázala struktura, která je uvedena ve zjednodušené podobě na obrázku 8. Tento návrh je založen na vzoru *supertyp/subtyp* (Hay, 2013). Jako supertyp zde vystupuje obecná položka v podobě tabulky `items`. Obsahuje atributy, které jsou společné pro všechny požadované typy položek. Na tuto tabulkou pak mají referenci konkrétní položky (například články, fotoalba, události apod.), které jsou v roli subtypů. Ty mají tedy nejen vlastní atributy a vazby, ale také všechny atributy a vazby, které má tabulka `items` (protože je zde povinná reference na tuto tabulku). ER diagram databáze pro konkrétní instanci systému je uveden v příloze B.



Obrázek 8: Finální návrh struktury databáze.

Tento vzor může být v objektově orientovaném programování implementován jako dědičnost. Toho lze využít při tvorbě modelových tříd aplikace. Rodičem zde bude třída `ItemsModel`, která umožní DML operace nad tabulkou `items`. Tato třída bude abstraktní a jakýkoliv třída, která bude představovat subtyp položky, pak musí dědit z této třídy. Tímto způsobem je možné předejít tomu, aby v databázi vznikl záznam samotné obecné položky.

Určitý problém u tohoto řešení představuje vazba 1:1 mezi konkrétní a obecnou položkou. Je třeba zajistit, aby nedošlo vytvoření více konkrétních položek s referencí na jednu položku obecnou. To však představuje daleko menší režijní nároky, než tomu bylo u EAV modelu.

Při pohledu na tabulky `items` a `pages` je zřejmá analogie s navrženými presentery `ItemPresenter` a `PagePresenter` modulu `FrontModul`. Právě tyto presentery vyžadují pro svůj chod primárně záznamy z těchto tabulek. Vzhledem k důležitosti těchto tabulek tedy bude uveden popis jejich atributů. Nejdříve tabulka `items`:

- `slug` – nepovinný atribut, který však musí mít vyplněny všechny položky, které mohou být otevřeny na samostatné stránce. Musí být unikátní.
- `title` – název dané položky. Obsah `<title>` elementu stánky. Podobně jako slug i title musí mít ty položky, které lze samostatně otevřít. Nepovinný atribut.
- `comments_allowed` – umožňuje povolit/zakázat vkládání komentářů.
- `date_created` – datum vytvoření položky.
- `date_updated` – datum poslední úpravy. Nepovinný atribut.
- `id_users` – odkazuje na autora položky.
- `id_pages` – reference na stránku, na které byla položka vytvořena.
- `id_item_types` – reference na typ položky. Název typu položky je shodný s názvem tabulky, ve které jsou položky tohoto typu uloženy. Z tabulky `item_types` lze také vyčíst, jestli je možné daný typ položek otevřít na samostatné stránce. Tuto informaci poskytuje hodnota atributu `is_linkable`.

Popis navržených atributů pro tabulku `pages`:

- `slug` – název stránky v URL. Musí být unikátní.
- `title` – název dané stánky. Obsah `<title>` elementu stránky. Unikátní.
- `menu_title` – název, pod kterým stránka figuruje v menu. Unikátní.
- `sort` – určuje pořadí v menu.
- `homepage` – informuje, jestli jde o domovskou stránku.
- `in_menu` – ovlivňuje viditelnost názvu stránky v menu.
- `disabled` – stanovuje, jestli je stránka přístupná, nebo nikoliv. Stránky tedy nemůže uživatel prostřednictvím redakčního systému smazat, ale pouze deaktivovat.
- `id_pages_type` – reference na typ stránky.

Závěrem této podkapitoly shrnu důvody pro volbu tohoto návrhu. Hlavní pozitiva jsou tedy přehlednost, soustředění společných atributů na jednom místě, relativně

snadná rozšiřitelnost v kombinaci s nižšími nároky na režii a jednoduchost SQL dotazů (v porovnání s EAV modelem).

4.4 Integrace webových služeb

Vyvíjený redakční systém má být propojen s webovými službami, které poskytuje Facebook, Google a Twitter. Je tedy potřeba, aby redakční systém získal přístup a práva k provádění operací v rámci těchto služeb. Ačkoliv jde o služby od tří odlišných společností, přístup k jejich API je založen na stejné myšlence – využívají totiž *OAuth* protokol.

Cílem tohoto protokolu je definovat jednotná pravidla, která zajistí bezpečné zasílání autorizovaných požadavků na různé webové služby. Výhodou této metody autorizace je, že uživatel nemusí zadávat svoje přihlašovací údaje na straně klientské aplikace, což přispívá k vyšší bezpečnosti (Boyd, 2012). Musí se ovšem přihlásit v aplikaci poskytovatele služby.

Na konci úspěšného autorizačního procesu obdrží klientská aplikace tzv. *přístupový token*. Tento objekt si lze představit jako pověření, které klientské aplikaci umožní zasílat požadavky na danou webovou službu jménem uživatele, jenž zmíněný proces absolvoval.

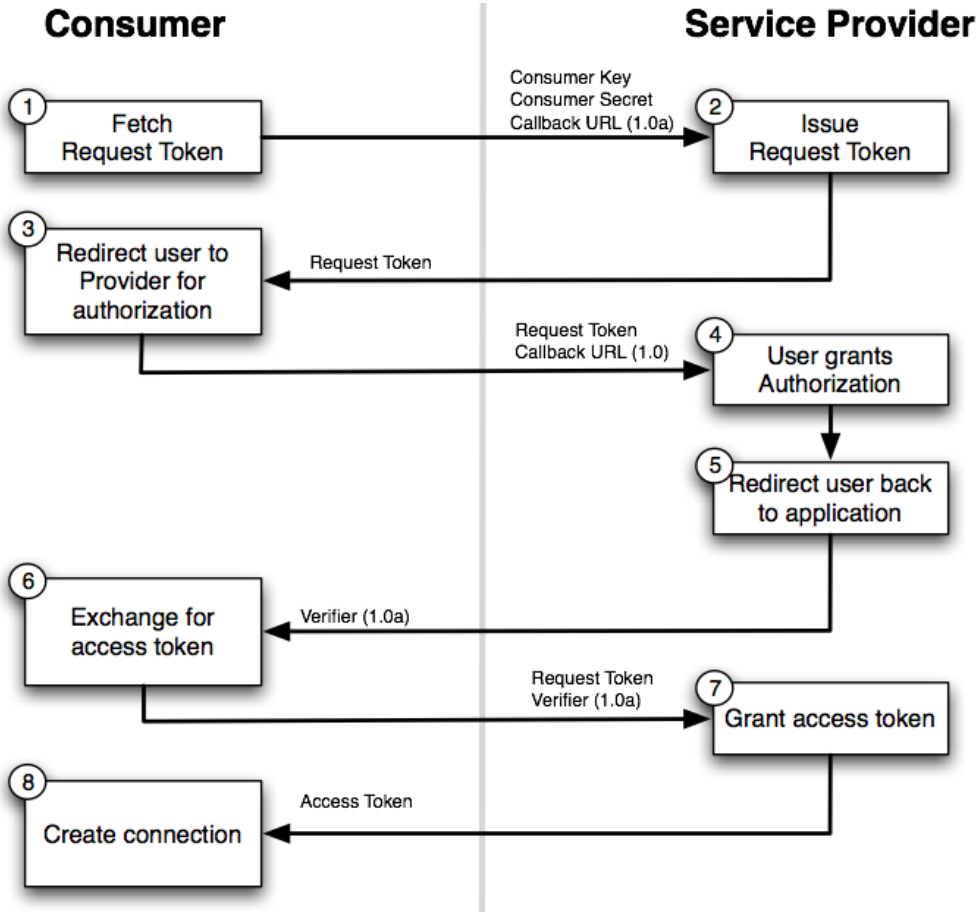
4.4.1 Získání přístupových tokenů

Jak bylo zmíněno výše, všechny tři společnosti vychází ze standardu OAuth. Twitter však implementuje verzi 1.0a[5], zatímco Facebook a Google verzi 2.0, která není zpětně kompatibilní (IETF, 2012). Jako nejzásadnější rozdíly mezi těmito verzemi uvádí Hammer (2010) tyto:

- Přístupové tokeny v OAuth 2.0 mají oproti předchozí verzi výrazně kratší životnost. Po jejich expiraci je však lze obnovit s využitím tzv. *refresh tokenu*.
- OAuth 2.0 usnadňuje odesílání požadavků zavedením tzv. *bearer tokenu*. Zatímco v předchozí verzi bylo nutné v HTTP `Authorization` záhlaví zadat několik parametrů, nyní stačí uvést pouze jediný řetězec reprezentující bearer token.
- Bearer token nevyžaduje šifrování na straně klientské aplikace. Aplikace však nově musí komunikovat s API pouze přes HTTPS.
- OAuth 2.0 doporučuje používat zvláštní server pro autorizaci uživatele a obsluhu požadavků na API.
- Verze 2.0 také nově zavádí postupy, díky kterým může uživatel provést autorizaci také přímo v desktopových a mobilních aplikací.

[5]Twitter sice umožňuje také OAuth 2.0 autorizaci, ta je však určená pro registrovanou aplikaci (ne uživatele), takže nabízí omezené možnosti (Twitter, 2017a).

Na obrázcích 9 a 10 jsou znázorněny kroky procesu, který vede k získání přístupových tokenů. Při srovnání těchto obrázků si lze všimnout rozdílů ve způsobu komunikace mezi *aplikací* (Consumer), která vyžaduje přístupový token a *poskytovatelem služby* (Service Provider) v závislosti na použité verzi OAuth.



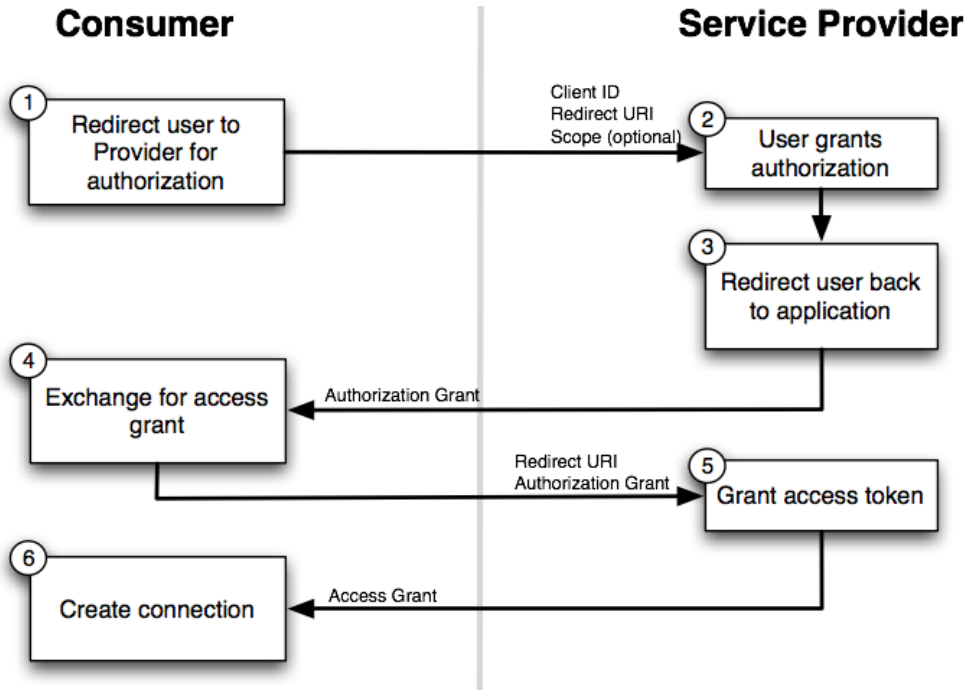
Obrázek 9: OAuth 1.0 flow diagram.
Zdroj: Walls a kol., 2014.

Průběh úspěšné autorizace u OAuth 1.0 je možné shrnout do tří fází:

1. Aplikace vyžádá a následně obdrží neautorizovaný request token (kroky 1 a 2).
2. Uživatel je přesměrovaný na autorizační stránku poskytovatele (3), kde obdržený request token autorizuje (4), načež je přesměrován zpět do aplikace (5).
3. Aplikace vymění ověřený request token za přístupový token (kroky 6 až 8).

Při porovnání obou verzí OAuth je patrné, že u verze 2.0 odpadá první zmíněná fáze. Namísto ověřeného request tokenu pak aplikace obdrží v kroku 3 autorizační kód a ten následně vymění za přístupový token, se kterým získá i refresh token.

Tyto rozdíly tedy nejsou natolik výrazné, aby nebylo možné vytvořit jednotné rozhraní pro autorizaci u všech tří poskytovatelů. Více v kapitole 4.4.4.



Obrázek 10: OAuth 2.0 flow diagram.
Zdroj: Walls a kol., 2014.

4.4.2 Vlastnosti přístupových tokenů

Přístupový token v sobě nese více údajů, než jen identitu uživatele, který stojí za jeho vznikem. V této podkapitole jsou uvedeny informace o nejdůležitějších vlastnostech, které byly brány v potaz při návrhu společného rozhraní.

Jednou z podstatných vlastností, které se k přístupovým tokenům váží, jsou přidělená přístupová práva. Rozsah přístupových práv, která bude aplikace vyžadovat, by měl být vždy nejmenší potřebný. Pokud bude záměrem pouze ověřit identitu uživatele za účelem jeho přihlášení do dané aplikace, větší rozsah požadovaných práv, než je získání uživatelova emailu, by mohl uživatele odradit od dokončení tohoto procesu.

Způsob získávání přístupových práv se liší v závislosti na poskytovateli:

- Facebook – jak je vidět na obrázku 10, rozsah práv (neboli scope) je jedním z parametrů autorizačního požadavku. O změnu rozsahu práv je možné zažádat kdykoliv to bude nutné. Uživatel však musí tuto změnu potvrdit.
- Google – tu stejnou možnost jako Facebook nabízí i Google. Změnu rozsahu pravomocí lze dosáhnout konfigurací třídy `Google_Client`, a to povolením tzv. *inkrementální autorizace* (Google, 2017a).
- Twitter – rozsah práv je možné měnit pouze v rámci webových stránek Twitteru, kde je aplikace zaregistrovaná. Ze strany aplikace zaslat požadavek o rozšíření práv nelze.

Dalším významným faktorem je životnost tokenu. V případě životnosti jsou patrné určité odlišnosti nejen v závislosti na poskytovateli, ale i účelu tokenu:

- Facebook – zde jsou dvě varianty:

  1. Token pro přístupová práva k uživatelskému účtu – token expiruje za 60 dní, pak je nutné přimět uživatele, aby autorizační proces zopakoval.
  2. Token pro přístupová práva ke stránce, kterou uživatel spravuje – tento token má neomezenou životnost (Facebook, 2017b).

  Přístup, který uplatňuje Facebook, je tedy evidentně v rozporu se standardem OAuth 2.0, poněvadž nabízí přístupový token s neomezenou životností namísto možnosti obnovy tokenu pomocí refresh tokenu.

- Google – ve výchozím nastavení je životnost tokenu omezená na 60 minut. Pokud však při konfiguraci třídy `Google_Client` nastavíme typ přístupu na *offline*, získáme kromě přístupového tokenu také refresh token, který umožní přístupový token neomezeně dlouho obnovovat (Google, 2017a).
- Twitter – poskytovaný token nikdy neexpiruje (Twitter, 2017b).

Obdržený token tedy může mít buď neomezenou životnost, nebo krátkou životnost, ale je možné jej neomezeně obnovovat. Proto je vhodné tyto tokeny někde uchovávat pro další použití. To by ovšem mohlo vést k mylnému dojmu, že jednou získaný token poskytuje nezvratnou doživotní možnost přístupu k potřebným zdrojům. Stále je však nutné kontrolovat, jestli je token aktivní. Protože se může snadno stát, že uživatel, který prošel autorizační proces, práva přidělená naší aplikaci (třeba i nechtěně) zruší.

4.4.3 Uchovávání tokenů

Jelikož je potřeba přístupové tokeny dlouhodobě využívat, budou uchovány v databázi. Přesněji v tabulkách zachycených na obrázku 11. Tokeny všech tří poskytovatelů jsou tvořeny z odlišných údajů, proto byly rozděleny do zvláštních tabulek.

Obrázek 11: Tabulky pro přístupové tokeny.

Je důležité si uvědomit, že přístupové tokeny představují možnost přístupu k operacím v rámci uživatelských účtů. Pokud by se tedy neoprávněné osobě podařilo získat přístup k databázi, kde jsou tokeny uloženy, získá tím veškeré pravomoci, které jsou s nimi spojené.

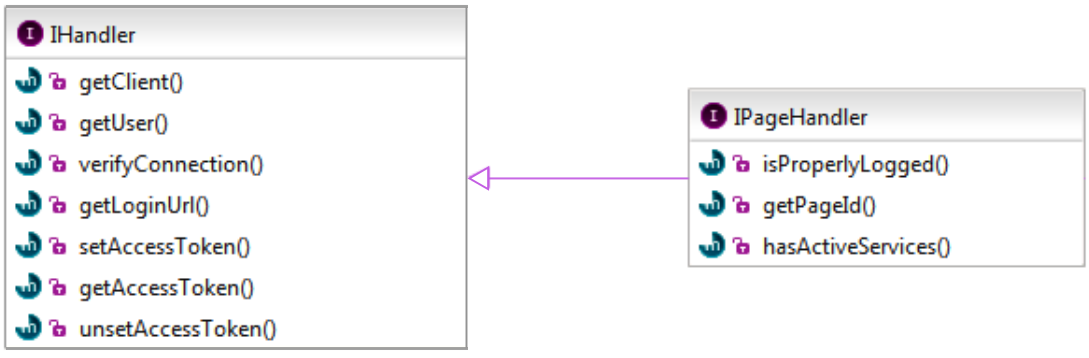
Toto bezpečnostní riziko je možné omezit šifrováním tokenů. K tomuto účelu byla vybrána knihovna *php-encryption*, s jejíž pomocí lze provádět symetrické šifrování. Klíč, který bude pro šifrování a dešifrování použit, bude uchováván v konfiguračním souboru. Útočník by tedy musel získat přístup nejen k databázi, ale také k tomuto souboru. Tím se riziko zneužití tokenů snižuje. Podrobnosti k implementaci šifrování s využitím této knihovny jsou uvedeny v podkapitole 5.2.2.

Zašifrovaný řetězec představující přístupový token, refresh token a token secret (to znamená údaje, které by mohl útočník zneužít) pak bude ukládán do sloupce s datovým typem BLOB. Volbu tohoto datového typu v těchto případech doporučuje ve své dokumentaci MySQL (Oracle Corporation, 2017).

4.4.4 Společné rozhraní pro autorizaci

Důvodem návrhu společného rozhraní byla snaha sjednotit způsob autorizace napříč různými poskytovateli. Ačkoliv je totiž postup při získávání přístupových tokenů u všech poskytovatelů srovnatelný (viz podkapitola 4.4.1), jejich knihovny (které budou v této práci využívány pro komunikaci s požadovanými API) mají odlišnou implementaci.

Společné rozhraní tedy umožní volat vždy ty samé metody, ať už bude třeba zprostředkovat autorizaci u Facebooku, Googlu nebo Twitteru. Tyto metody jsou zachyceny na obrázku 12.



Obrázek 12: IHander a IPageHandler.

Z obrázku 12 je také patrné, že byla navržena dvě rozhraní – `IHandler` a jeho potomek `IPageHandler`. Bude totiž potřeba provádět autorizaci pro dva různé typy

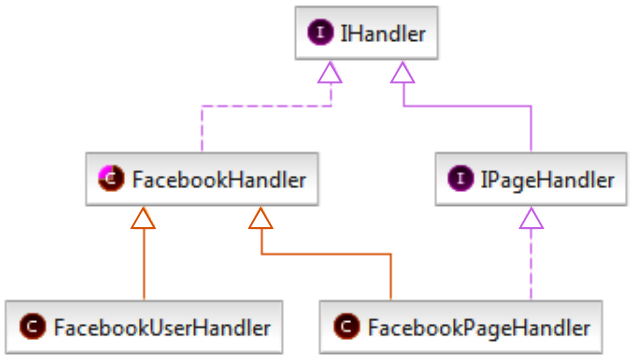
účtů[6]:

1. Uživatelský účet – získaný přístupový token bude použitý pouze pro ověření identity uživatele. Tedy při přihlašování do systému (viz následující podkapitola).
2. Účet představující stránku – to znamená účet, který má být propojen s danou instancí systému. Jestli jde o tento typ účtu, bude ověřeno při autorizaci porovnáním ID účtu, které předá API, s ID nastaveným v konfiguračním souboru.

Autorizace pro první typ účtu bude prováděna pomocí tříd, které implementují `IHandler`. Jak tedy vyplývá, třídy implementující `IPageHandler` jsou navrženy pro druhý uvedený typ. Všechny třídy implementující tato rozhraní budu pro zjednodušení dále označovat jako *handlery*.

Hierarchie handlerů je však ještě o něco složitější. Jak je patrné z obrázku 13, rozhraní `IHandler` implementují všechny uvedené třídy. První třída v hierarchii je abstraktní `FacebookHandler`, který obsahuje obecně použitelné metody. Implementace potomků této třídy se pak liší podle požadovaných přístupových práv a způsobu manipulace s přístupovými tokeny.

`FacebookUserHandler` vyžaduje pouze přístup k emailu uživatele a token nikde neuchovává. Naproti tomu `FacebookPageHandler` bude požadovat všechna práva pro provádění požadovaných operací s danou Facebookovou stránkou a obdržený token bude ukládat do databáze. Navíc implementuje `IPageHandler`, takže nabízí větší rozsah funkcionality.



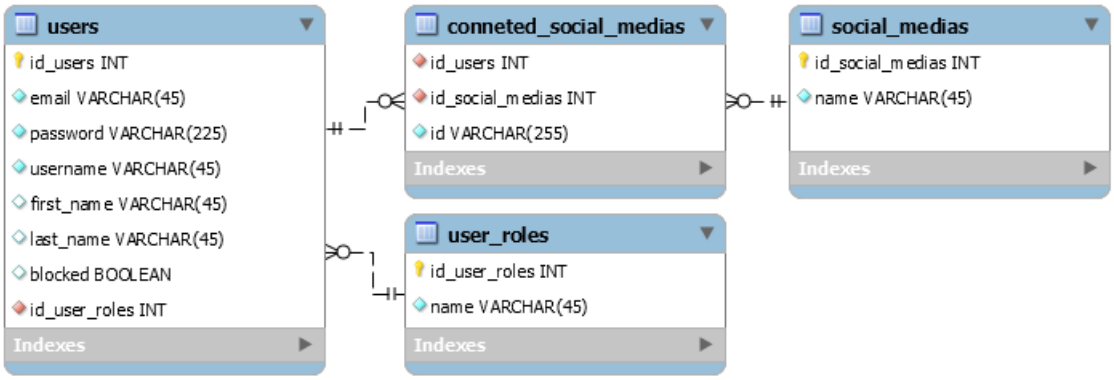
Obrázek 13: Handlery pro autorizaci uživatele na Facebooku.

Třídy typu `*UserHandler` a `*PageHandler` budu dále v textu označovat jako *user handlery* a *page handlery*. Možné scénáře použití obou těchto variant handlerů popisují následující podkapitoly.

[6]Toto rozdělení zohledňuje způsob, jakým jsou účty chápány v rámci redakčního systému. Z pohledu poskytovatele může jít (a z pravidla také půjde) o ten samý typ účtu.

4.4.5 Přihlašování přes sociální sítě

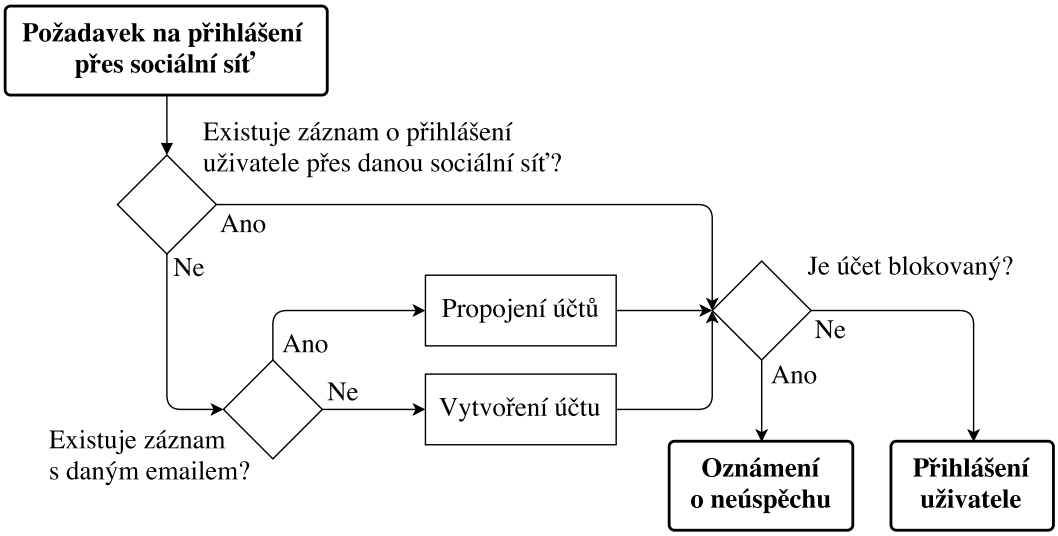
Tou úplně nejzákladnější operací, kterou je možné po získání přístupového tokenu (s využitím příslušného user handleru) provést, je přihlášení nebo případně registrace uživatele. Jakmile aplikace obdrží token, zašle na příslušné API dotaz na identitu uživatele. Vždy bude požadovat uživatelovo ID, jméno a emailovou adresu.



Obrázek 14: Databázové tabulky související s uživatelským účtem.

S prvním přihlášení přes sociální síť dojde také k registraci uživatele. S tím je spojeno vytvoření záznamu v tabulce `users` a `connected_social_medias` (viz obrázek 14). V tabulce `users` jsou uloženy vyžádané informace o uživateli vyjímaje uživatelovo ID, které je vloženo do tabulky `connected_social_medias`. Na základě tohoto ID dokáže tedy systém při dalším přihlašování přiřadit uživateli ten správný účet.

Pro uživatele, kteří účet na sociální síti nemají, nebo se tímto způsobem nechtějí přihlašovat, existuje druhá varianta registrace a přihlašování. Ta vyžaduje vyplnění příslušných formulářů, kde je mimo jiné vyžadována i emailová adresa.



Obrázek 15: Vývojový diagram přihlašování přes sociální sítě.

Návrh přihlašování přes sociální sítě počítá s oběma variantami vytvoření účtu. Přihlašovací proces znázorňuje vývojový diagram na obrázku 15.

V první fázi je tedy ověřeno, jestli existuje záznam s uživatelovým ID v tabulce `connected_social_medias`, na základě kterého by systém přiřadil uživateli jeho účet. Pokud zde takový záznam není, je zřejmé, že se uživatel přes danou sociální síť ještě nepřihlásil. Je však třeba ověřit, zda se tento uživatel neregistroval druhým zmíněným způsobem (prostřednictvím formuláře).

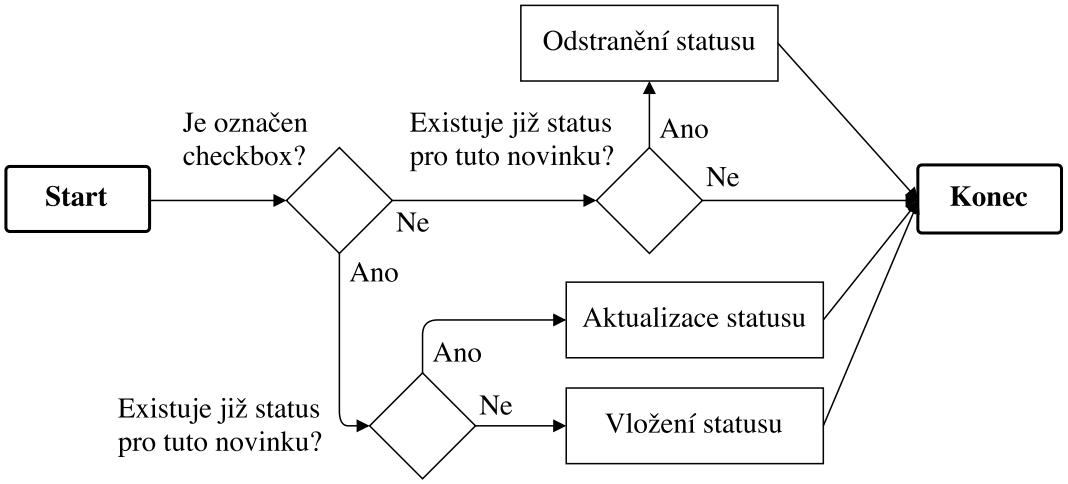
Systém se tedy pokusí vyhledat v tabulce `users` uživatelský účet s emailem, který obdržel při dříve zmíněném dotazu na identitu uživatele. Pokud je takový účet nalezen, systém vytvoří v tabulce `connected_social_medias` nový záznam s uživatelovým ID, čímž je uživatelský účet v redakčním systému spárován s účtem na sociální síti. Pokud uživatelský účet s daným emailem nalezen není, systém uživatele zaregistruje.

V posledním kroku je ještě potřeba ověřit, jestli nebyl požadovaný uživatelský účet zablokován. O tom informuje hodnota ve sloupci `blocked` v tabulce `users`. Pokud je tato hodnota negativní, dojde k přihlášení uživatele. Jinak je uživateli oznámeno, že byl jeho účet zablokován.

4.4.6 Vkládání statusů na sociální sítě

Jednou z pokročilejších funkcí, kterou je možné uživateli usnadnit správu jeho stránek, je automatizované vkládání statusů na sociální sítě spjaté s danou instancí redakčního systému. V okamžik, kdy uživatel vloží na svůj web například novinku, může zároveň upozornit o jejím zveřejnění také uživatele na Facebooku nebo Twitteru.

Google umožňuje vkládání statusů pouze s využitím *Google+ Domains API* (Google, 2017b). Toto API je však použitelné jen pro účty, které jsou součástí placeného řešení *G Suite* (Google, 2017c).



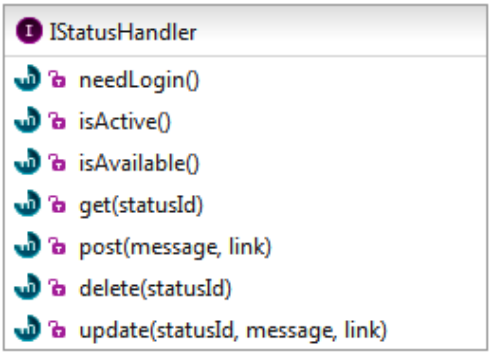
Obrázek 16: Vývojový diagram vkládání a úpravy statusu.

Volbu této možnosti lze zajistit prostřednictvím checkboxu ve formuláři pro vkládání novinky. Ten samý formulář bude používaný i pro úpravu novinek. Proto bylo nutné navrhnout, jak vyřešit situace, které mohou nastat při odeslání zmíněného formuláře. Tyto situace zachycuje vývojový diagram na obrázku 16.

Je tedy třeba brát úvahu, zda je označený checkbox a také, jestli už status pro tuto novinku nebyl vložen. Po vložení statusu na sociální síť (či sítě) proto musí být uloženo ID tohoto statusu (případně statusů) do databáze. Ke vložení, aktualizaci nebo odstranění statusu pak dojde v závislosti na (ne)existenci těchto identifikátorů v záznamu a (ne)označení checkboxu.

Podobně jako u autorizace, i zde bude prováděn stejný typ operace u různých poskytovatelů služeb. Proto bylo vytvořeno jednotné rozhraní také pro třídy, které budou realizovat vkládání statusů. Jak je vidět na obrázku 17, kromě CRUD operací je zde také metoda `isActive()` určená ke zjištení, zda je přidávání statusů v dané instanci systému povoleno. Tato informaci bude získávána z konfiguračního souboru.

Metoda `isAvailable()` pak vrací, jestli existuje přístupový token v lokální databázi a `needLogin()` umožňuje kontrolovat, zda je token aktivní na straně poskytovatele. Samozřejmě by šlo používat pouze druhou zmíněnou, odpověď na tento požadavek však trvá poměrně dlouhou dobu. Tato metoda je tedy navržena pro použití až při vkládání statusu. V ostatních případech je vhodnější využívat `isAvailable()`.



Obrázek 17: Rozhraní pro handlery, které umožňují práci se statusy na sociálních sítích.

Stejným způsobem je navrženo i vkládání událostí do Google kalendáře.

4.4.7 Stahování fotoalb a událostí z facebooku

Tato funkcionalita je navržena poněkud odlišným způsobem. Nejdříve je nutné události či alba vytvořit na Facebooku a až poté je nahrát na svůj web. Lze tedy mluvit o synchronizaci, ale pouze v jednom směru. Facebook totiž neumožňuje vytváření událostí přes API. Nahrávání fotoalb sice toto API umožňuje, ale z hlediska jednotnosti jsem se rozhodl řešit synchronizaci alb i událostí stejným způsobem.

V obou případech může synchronizace probíhat automaticky, anebo s interakcí uživatele. Ten si pak může zvolit, která alba či události na svých stránkách chce. Zda

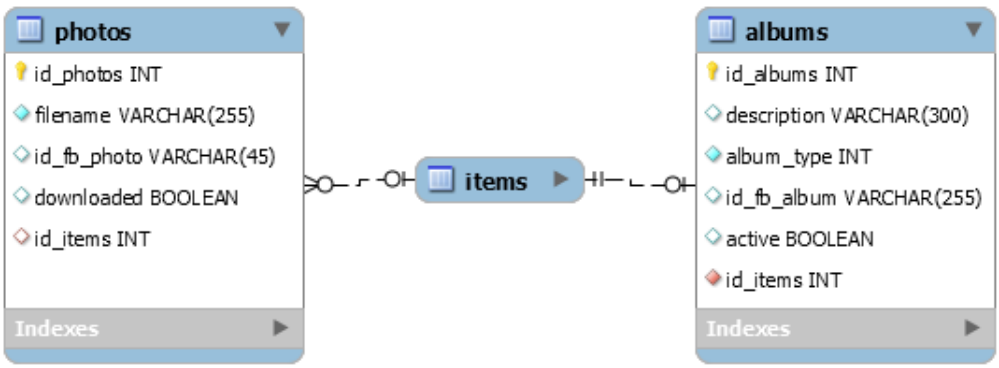
bude tento proces probíhat automaticky nebo ne, bude nastaveno v konfiguračním souboru.

Zatímco synchronizace událostí představuje pouhé stažení dat ve formátu JSON, synchronizace alb navíc vyžaduje také stažení obrazových souborů (ve formátu JPEG). Tato operace je časově velmi náročná. Proto je nevhodné po uživateli chtít, aby na stránce počkal, než dojde k provedení této operace. Zaprvé by tím došlo k omezování uživatele, zadruhé by tak pravděpodobně stejně nebylo dosaženo požadovaného výsledku. To proto, že může být v průběhu stahování překročen časový limit pro provedení skriptu[7].

Automatickou synchronizaci i problém se stahováním fotoalb je možné vyřešit s použitím Cronu. Při synchronizaci událostí je nutné si přes API Facebooku vyžádat všechny potřebné údaje a ty pak společně s ID události vložit do příslušné databázové tabulky. S každým voláním Cronu je tedy porovnáním ID prověřeno, jestli na Facebooku nepřibyla nová událost nebo nedošlo ke změně v události, která byla dříve stažena. Na základě toho pak dojde k vložení či aktualizaci hodnot.

Stahování fotoalb však představuje větší komplikaci. Cron totiž bude spouštěn v určitém intervalu, takže je nutné stahování obrázků přerušit vždy před dalším spuštění Cronu. Jinak by mohlo dojít ke souběžnému spuštění té samé operace a systém by mohl stahovat vícekrát ty samé obrázky.

Aby k této situaci nedocházelo, je nutné nastavit časový limit pro stahování obrázků. Například v konfiguračním souboru. Tento limit musí představovat nižší hodnotu, než jaký je časový interval mezi spouštěním Cronu.



Obrázek 18: Tabulky pro vkládání fotoalb.

V tabulce `photos` (viz obrázek 18) má důležitou roli sloupec `downloaded`, který signalizuje, jestli již došlo ke stažení obrázku. Tabulka `albums` zase obsahuje sloupec `active`, který informuje o tom, jestli je dané album určeno ke stažení, či nikoliv. Při každém spuštění Cronu jsou tedy vždy vyžádány obrázky, které mají nepravdivou hodnotu sloupce `downloaded` a pravdivou hodnotu sloupce `active`.

[7] Časový limit pro provedení skriptu je určen parametrem `max_execution_time`. Tento limit by neměl nabývat vyšších hodnot než několika jednotek minut.

Synchronizaci je možné provádět ve dvou režimech:

1. Iniciálně – tuto variantu je nutné zvolit při zavedení systému, který bude propojen s Facebookovou stránkou, kde již existují výše uvedené položky.
2. Inkrementálně – na rozdíl od iniciálního režimu stahuje především nově vzniklé záznamy. Kvůli možným změnám v již stažených záznamech vyžaduje od API také záznamy starší, u kterých kontroluje datum poslední úpravy. Pokud k úpravě došlo, dojde k aktualizaci záznamu v databázi.

5 Implementace

Pro ukázku výsledku implementace budou v následujících podkapitolách uvedeny úryvky kódu a snímky webu *Police Symphony Orchestra*, tedy jednoho z webů, na kterých byl vyvíjený redakční systém nasazen.

5.1 Implementace redakčního systému

V následujících podkapitolách je popsána implementace částí systému, které jsou nejdůležitější z hlediska nasazování nových instancí.

5.1.1 Konfigurace

Standardně se v Nette provádí konfigurace v souborech typu NEON[8]. V této aplikaci je konfigurační soubor `config.neon` umístěn v adresáři `app/config`. Účelem tohoto souboru je nastavit hodnoty, které jsou specifické pro danou instanci systému. Jde především o konfiguraci připojení k databázi, registraci objektů, které daná instance systému vyžaduje ke své činnosti (tzv. služeb) a nastavení hodnot parametrů, které mají být těmto službám předány.

Zmíněné typy konfigurace jsou v konfiguračním souboru členěny do tří různých sekcí – `database`, `parameters` a `services`. Nastavení databáze může vypadat například takto:

```
database:
    driver: mysql:host=127.0.0.1;dbname=pso
    username: root
    password:
```

Jde o výchozí způsob konfigurace databázového připojení. Předávání těchto hodnot službě, která toto připojení zajišťuje, provádí framework. Vlastní parametry, které je potřeba předat například komponentám z důvodu customizace, je možné nadefinovat v sekci `parameters`:

```
parameters:
    contactEmails:
        supportMail: support@example.com
        adminMail: admin@example.com
```

Předání těchto parametrů musí zajistit implementátor. Nejjednodušší cestou je předání do konstruktoru třídy, která je zaregistrovaná jako služba v sekci `services`:

```
services:
    - App\SocialMedia\Auth\LoginManager(%contactEmails%)
```

[8]NEON je zkratkou pro *Nette Object Notation* (GitHub, 2017b).

Třída `LoginManager` vyžaduje ve svém konstruktoru kromě výše uvedeného parametru také další služby. Ty zde však není třeba explicitně uvádět. Dosazení požadovaných služeb totiž zajišťuje dependency injection (DI).

Odsazování řádků, které je v uvedených příkladech vidět, není jen otázkou zpřehlednění konfigurace. Tento způsob zápisu je vyžadován z hlediska NEON syntaxe. Na základě odsazování řádků v uvedeném příkladu sekce `parameters` dokáže třída `Nette\Neon\Decoder`, která je určena k parsování konfiguračního souboru, vytvořit toto pole:

```
array (
    "contactEmails" => array (
        "supportMail" => "support@example.com"
        "adminMail" => "admin@example.com"
    )
)
```

Kromě tří uvedených sekcí obsahuje konfigurační soubor také další sekce, které se týkají nastavení aplikace. Tato nastavení jsou však společná pro všechny instance systému. Při vytváření nové instance se v nich nic měnit nebude. Zkrácená verze konfiguračního souboru je k vidění v příloze C.

Podrobnější příklad předání parametru z konfiguračního souboru do registrované služby je uveden v podkapitole 5.2.1.

5.1.2 Společné komponenty

Jak již bylo řečeno v návrhu, komponenta je základním prvkem aplikace v Nette. Použité komponenty tedy určují podobu i funkcionalitu redakčního systému. Některé komponenty systému jsou společné pro každou instanci. Jde o komponenty, které jsou umístěny ve jmenném prostoru `App\FrontModule\Components\Common`.

Patří sem například menu pro výběr sekcí v záhlaví webu, formuláře pro přihlašování a vkládání komentářů a také rozhraní pro správu sekcí a uživatelů aplikace. Toto rozhraní je dostupné pouze pro implementátora. Více informací o úloze tohoto rozhraní je uvedeno v podkapitole 6.1.

5.1.3 Implementace komponent

Tato kapitola se věnuje komponentám, které jsou specifické pro danou instanci systému. Ty se nachází ve jmenném prostoru `App\FrontModule\Components\Custom`. Protože se instance systému liší právě tím, jaké prezentuje položky, patří sem zejména komponenty, které umožňují CRUD operace nad položkami. S každým přidaným typem položky musí implementátor absolvovat tyto kroky:

1. Vytvoření databázové tabulky, která bude reprezentovat daný typ položky. Tato tabulka musí být subtypem tabulky `items` (viz podkapitola 4.3.3).

2. Vytvoření modelové třídy `*Model` ve jmenném prostoru `App\Model\Item`.
3. Implementace tříd (respektive komponent) `*FormFactory` a `*Control` ve zmíněném jmenném prostoru `App\FrontModule\Components\Custom`.
4. Vytvoření šablony `*.latte`, která představuje front-end podobu komponenty.
5. Vložení komponent do šablon presenterů.

Na místě hvězdičky bude konkrétní název položky. Tento název se shoduje s názvem tabulky, ve které jsou záznamy týkající se daného typu položky. Všechny výše uvedené třídy musí být v konfiguračním souboru zaregistrovány jako služby.

V případě webu Police Symphony Orchestra jsou položkami například členové orchestru. Pro tento typ položky byla vytvořena v databázi tabulka `musicians`. Z toho vyplývá, že bylo nutné vytvořit třídu `App\Model\Item\MusiciansModel`. Tato třída, tak jako všechny modely operující nad tabulkami se záznamy o položkách, musí být potomkem třídy `App\Model\Item\BaseItemsModel`.

`BaseItemsModel` obsahuje metody pro vkládání, úpravu a odstraňování záznamů tabulky `items`. Protože je mezi tabulkou `items` a `musicians` povinná vazba, jejich záznamy musí vznikat i zanikat současně. Metody třídy `BaseItemsModel` proto musí být volány v rámci stejného typu metod potomka `MusiciansModel`.

Obrázek 19: Formulář pro vkládání člena orchestru.

Dalším krokem byla implementace tovární třídy `MusiciansFormFactory`, která vytváří formulář pro vkládání a úpravu položek. Tento formulář tedy musí mít vstupy odpovídající sloupcům tabulky `musicians` a navíc také skryté vstupy pro ID člena

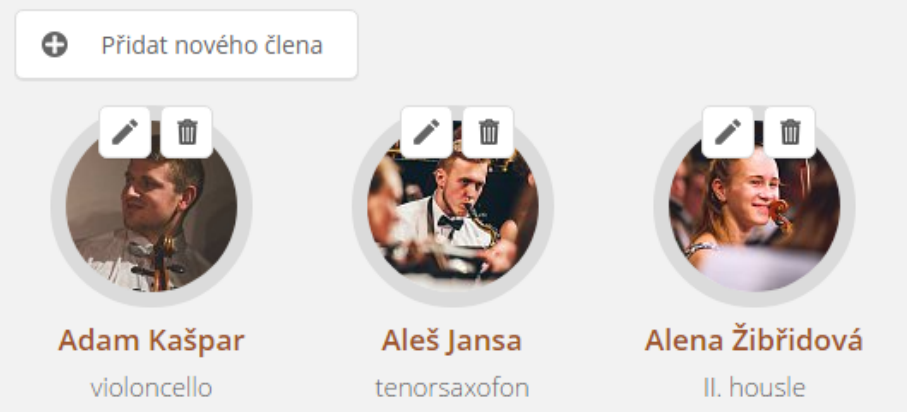
orchestru a stránky, na které je formulář umístěn. O vyplnění těchto hodnot se stará třída `MusiciansControl` (viz níže).

Třída `MusiciansFormFactory` vyžaduje ve svém konstruktoru předání modelu `MusiciansModel`. Příslušné metody tohoto modelu totiž využívá při vkládání a úpravě záznamů. K úpravě záznamu dojde, pokud je vyplněna hodnota ID člena orchestru. Vizuální stránku formuláře zachycuje obrázek 19.

V další fázi byla vytvořena třída `MusiciansControl`, která zmíněný formulář využívá. Vyžaduje ho tedy v konstruktoru, stejně tak jako třídu `MusiciansModel`. Metody třídy `MusiciansControl` mimo jiné zajišťují zobrazení a předvyplňování hodnot do vstupů zmíněného formuláře:

- `createComponentMusicianForm()` – vytvoří instanci formuláře.
- `render()` – vykreslí šablonu `musicians.latte` s výpisem členů orchestru.
- `handleAdd()` – zobrazí prázdný formulář určený k přidání nového záznamu se členem orchestru (viz obrázek 19) a předvyplní ID stránky.
- `handleUpdate($idMusicians)` – zobrazí formulář a předvyplní jeho vstupy údaji o členovi orchestru na základě předaného ID.
- `handleDelete($idMusicians)` – smaže záznam z databáze.
- `handleSort($idMusicians, $order)` – změní pořadí člena orchestru.

Na obrázku 20 jsou vidět prvky administračního rozhraní, které volají výše uvedené metody. Tlačítko *Přidat nového člena* volá metodu `handleAdd()`. Tlačítko s ikonkou tužky (vlevo nahoře v profilovém obrázku člena orchestru) zprostředkovává metodu `handleUpdate()` a tlačítko s ikonkou koše (vpravo nahoře) zajišťuje mazání záznamu metodou `handleDelete()`. Metoda `handleSort()` je využívaná při řazení způsobem drag and drop.



Obrázek 20: Výpis členů orchestru a prvky administračního rozhraní.

Podobu, kterou uvedená komponenta nabývá (viz obrázek 20), definuje šablona nastavená v metodě `render()` třídy `MusiciansControl`. V tomto případě jde o šablonu

`musicians.latte`. Šablony všech komponent jsou umístěny v adresáři `templates`, který je ve stejném adresáři, jako daná komponenta.

Takto vytvořené komponenty je třeba je umístit do šablon presenterů. Aby presenter věděl, ve které části stránky má komponentu vykreslit, musí být v jeho šabloně vloženo na požadovaném místě makro ve tvaru `{control názevKomponenty}`.

Vykreslení komponent, které vypisují položky, zajišťuje `PagePresenter`. Tento presenter má jedinou šablonu `default.latte`, která je umístěna v adresáři `templates\Page`. Následuje úryvek kódu této šablony, který zajistí vykreslení komponenty `MusiciansControl`:

```
{elseif $page->pageType === "orchestr"}

    {$page->title}

    {if $presenter->getView() === "admin"}
        <div class="admin-panel active">
            {control pageContentForm}
        </div>
    {else}
        {$page->content|noescape}
    {/if}

    {control musicians}
```

Tento kód představuje způsob, jakým je vyřešeno zobrazení stránky typu `orchestr`. Na předposledním řádku je uvedeno makro `{control musicians}`, které v šabloně reprezentuje komponentu `MusiciansControl`. Toto makro zavolá v presenteru metodu `createComponentMusicians()`, která musí vracet instanci zmíněné komponenty. Výchozí implementace presenterů Nette vyžaduje vytvoření metody tohoto typu pro každou komponentu. Takto by implementátor musel vytvořit presenter s odlišnou implementací pro každou instanci systému.

Abych této situaci předešel, přetížil jsem metodu `createComponent()`, která je za předávání komponent zodpovědná. Ta se místo volání zmíněné metody `createComponentMusicians()` pokusí najít třídu `MusiciansControl` v DI kontejneru. Vzhledem k tomu, že jsou všechny komponenty zaregistrovány jako služby, požadovaná komponenta musí být nalezena a implementátor je tak ušetřen od psaní zbytečného kódu navíc.

Stejným způsobem je tento problém vyřešen v presenteru `ItemPresenter`. Ten se stará o vykreslení samotného formuláře pro položky, které mohou být otevřeny na samostatné stránce. V takovém případě totiž není třeba komponenta, která vypisuje položky, protože je na stránce jediná položka daného typu. Pro každý takový typ položky je nutné vytvořit šablonu, jejíž název odpovídá názvu položky. Šablony tohoto presenteru jsou umístěny v adresáři `templates\Item`.

Šablona `musicians.latte` i třídy `MusiciansFormFactory` a `MusiciansControl` byly vytvořeny na základě koster, které představují soubory `items.latte.template`, `ItemsFormFactory.php.template` a `ItemsControl.php.template`. Tyto soubory jsou umístěné ve stejném adresáři, jako soubory s komponentami jmenného prostoru `App\FrontModule\Components\Custom`.

5.2 Integrace webových služeb

Tato podkapitola popisuje zásadní implementační kroky z hlediska integrace webových služeb.

5.2.1 Získání přístupových tokenů

Teoretický základ ohledně získávání přístupových tokenů, jejich vlastností a způsobu uchovávání je uveden v podkapitole 4.4. Cílem této podkapitoly je popsat, jaké kroky bylo třeba v tomto ohledu podniknout prakticky. Postup byl následující:

1. Registrace aplikace.
2. Vložení údajů získaných při registraci do konfiguračního souboru.
3. Implementace handlerů.
4. Vytvoření modelové třídy, která zajistí uchovávání získaných tokenů.

Prvním krokem tedy byla registrace aplikace na níže uvedených stránkách poskytovatelů. To znamená vyplnění formulářů požadovanými informacemi o aplikaci.

Vyžadované údaje se liší podle poskytovatele. Především je však nutné uvést název aplikace. V případě Googlu je nutná i adresa, na kterou má být uživatel přesměrován po autorizaci. Dále následují položky jako adresa, na které se web nachází, popis a ikona aplikace a odkaz na zásady ochrany soukromí a smluvní podmínky aplikace[9]. Registraci aplikace lze provést na těchto adresách (v závislosti na poskytovateli služby):

- Facebook – `https://developers.facebook.com/apps`
- Google – `https://console.developers.google.com`
- Twitter – `https://apps.twitter.com`

Po zadání požadovaných informací vygeneruje registrační aplikace dvojici údajů, které bude nutné předávat při požadavcích na API. Díky těmto údajům totiž dokáže server poskytovatele identifikovat, která z registrovaných aplikací s ním navazuje

[9]Odkaz na zásady ochrany osobních údajů a smluvní podmínky je nutný pouze u Twitteru, a to v případě, kdy klientská aplikace vyžaduje přístupová práva pro získání emailové adresy uživatele.

spojení (Boyd, 2012). Každý poskytovatel má pro zmíněné údaje jiné označení. V případě Facebooku jde o `App ID` a `App Secret`, u Google jde o `Client ID` a `Client secret` a Twitter má `Consumer Key` a `Consumer Secret`.

Registraci je nutné provést pro každou instanci systému. Získané údaje budou tedy odlišné pro každý web, na kterém bude systém nasazen. Nebudou proto vloženy přímo do kódu, ale do konfiguračního souboru, ze kterého budou předávány příslušným handlerům.

Dalším krokem byla implementace handlerů. Zatímco předchozí kroky je nutné provádět při nasazování každé instance systému, handlery jsou pro všechny instance stejné. S jejich implementací je spojeno stažení knihoven zmíněných tří poskytovatelů, které usnadňují zasílání požadavků na webové služby. Jde o tyto knihovny:

- Facebook SDK for PHP – knihovna pro *Graph API* (rozhraní Facebooku), která je dostupná na adrese: `https://github.com/facebook/php-graph-sdk`
- Google APIs Client Library for PHP – knihovna, která umožňuje využívat webové služby společnosti Google. Je dostupná na této adrese: `https://github.com/google/google-api-php-client`
- TwitterOAuth – Twitter oficiální knihovnu pro svoje API nemá, na svých stránkách však doporučuje tuto knihovnu Abrahama Williamse (Twitter, 2017c). Ta je dostupná zde: `https://github.com/abraham/twitteroauth`

Nejsnazším způsobem, jak tyto knihovny do projektu vložit, je použít Composer.

Každá z uvedených knihoven obsahuje jednu hlavní třídu, která je stěžejní pro veškeré operace. Pro Facebook je to třída `Facebook\Facebook`, pro Google `Google_Client` a Twitter `Abraham\TwitterOAuth\TwitterOAuth`.

Uvedeným třídám je nutné předat údaje získané při registraci aplikace. Tyto údaje jsou v konfiguračním souboru. Tam jsou také registrovány všechny handlery jakožto služby. Předání zmíněných údajů je tedy například pro `FacebookPageHandler` realizováno takto:

```
parameters:
    facebook:
        pageId: '1387327667997045'
        apiId: '221978694946369'
        secret: '689cf0a6a5a9475c96ce487dx6921ce3'
services:
    - App\SocialMedia\Auth\FacebookPageHandler(%facebook%)
```

Ve konstruktoru třídy `FacebookPageHandler` pak stačí deklarovat, že vyžaduje předání tohoto parametru. Tento parametr byl v konstruktoru pojmenován jako `$params` (může se ale jmenovat jakkoliv jinak, důležité je pořadí předávaných parametrů). Následující kód znázorňuje předání parametrů třídě `Facebook`, která je v konstruktoru zmíněného handleru instanciována:

```
$this->facebookClient = new Facebook([
    "app_id" => $params["apiId"],
    "app_secret" => $params["secret"],
    "default_graph_version" => "v2.8",
]);

if ($this->getAccessToken())
    $this->facebookClient->setDefaultAccessToken(
        $this->getAccessToken()
    );
```

V posledních čtyřech řádcích kódu je vidět metoda `getAccessToken()`. Jde o jednu z metod rozhraní `IHandler` uvedeného v podkapitole 4.4.4. Tato metoda vrací přístupový token. V uvedeném kontextu tedy umožní nastavit výchozí přístupový token pro danou instanci třídy `Facebook`, a tím pádem i odesílat požadavky na Graph API. Následuje popis ostatních metod rozhraní `IHandler`:

- `getClient()` – vrací instanci třídy `Facebook`.
- `getUser()` – Vrací pole s údaji o uživateli získané z API poskytovatele. Těmi jsou ID uživatele, jméno, příjmení, uživatelské jméno a email.
- `verifyConnection()` – ověřuje, jestli je k dispozici aktivní token. Tedy jestli například uživatel neodstranil práva pro danou aplikaci na straně Facebooku.
- `getLoginUrl()` – vrací adresu, na které má být uživatel autorizován.
- `setAccessToken()` – zajišťuje vygenerování a uložení přístupového tokenu.
- `unsetAccessToken()` – odstraní přístupový token.

Z hlediska autorizačního procesu (viz podkapitola 4.4.1) jsou uvedené metody využity tímto způsobem:

1. Uživatel klikne na odkaz, který vygeneruje metoda `getLoginUrl()` a je přesměrován na web poskytovatele.
2. Na webu poskytovatele provede autentizaci a autorizaci.
3. Uživatel je přesměrován zpět s autorizačním kódem v parametru adresy. Ten zpracuje metoda `setAccessToken()` a vytvoří přístupový token.
4. Po získání přístupového tokenu je možné využít metodu `getUser()` pro ověření identity uživatele.

V poslední uvedené ukázce konfiguračního souboru je také vidět parametr `pageId`. Ten vyžadují všechny page handlery. Jde totiž o identifikátor stránky (respektive účtu), která má být propojena s danou instancí systému (viz podkapitola 4.4.4).

Nasazování instance systému je tedy rovněž spojeno se získáváním těchto identifikátorů. Pokud však koncový uživatel takovou stránku nemá (a nechce mít), není třeba tento parametr vyplňovat.

Nejsnáze lze získat ID Google+ účtu. Identifikátor je u této služby součástí adresy uživatelského profilu. Nejjednodušším způsobem, jak obdržet ID účtu na Facebooku je využití služby, kterou nabízí web `findmyfbid.com`. Ten poskytne ID na základě adresy Facebookového profilu (či stránky). U Twitteru lze pro tento účel využít web `gettwitterid.com`, který vyžaduje zadání uživatelského jména daného účtu (to je součástí adresy požadovaného profilu).

Protože page handlery musí implementovat třídu `IPageHandler` (viz podkapitola 4.4.4), musí navíc obsahovat také tyto metody:

- `getPageId()` – vrací hodnotu parametru `pageId`.
- `isProperlyLogged()` – pravdivou hodnotu vrátí pouze tehdy, je-li dostupný přístupový token, a pokud ano, tak zda jsou s tímto tokenem svázána potřebná přístupová práva. Pokud ne, je to pro aplikaci signálem, že by měla přimět uživatele k provedení autorizace.
- `hasActiveServices()` – vrací, jestli je daný page handler nutné využít. To znamená, jestli je nutné získat token pro přístup ke službám daného poskytovatele.

Page handlery vyžadují ve svém konstruktoru kromě parametrů z konfiguračního souboru také službu `App\Model\TokenManager`. Jde o modelovou třídu, která zajišťuje všechny potřebné operace spojené s uchováváním přístupových tokenů v databázi. Mezi tyto operace patří také šifrování tokenů.

5.2.2 Šifrování tokenů

Jak bylo zmíněno v podkapitole 4.4.3, přístupové tokeny budou dlouhodobě uchovávány v databázi, což představuje bezpečností riziko. Proto budou řetězce, které je tokeny identifikují, šifrovány a v okamžiku, kdy mají být použity, opět dešifrovány. Pro tyto účely byla použita knihovna *php-encryption* (GitHub, 2017a).

Pro dosažení požadovaných výsledků, bylo nutné využít pět metod této knihovny. Na začátku je potřeba vygenerovat náhodný klíč, který bude při každém šifrování (a dešifrování) použit. K tomu slouží metoda `Key::createNewRandomKey()`. Nad takto vytvořeným klíčem musí být zavolána metoda `saveToAsciiSafeString()`, která převede klíč na řetězec tisknutelných znaků ASCII. Tento řetězec byl uložen do konfiguračního souboru, ze kterého je předáván do třídy `TokenManager`.

Jelikož třída `TokenManager` zajišťuje CRUD operace nad tabulkami s tokeny, dochází k šifrování a dešifrování přímo zde. Jak bylo uvedeno výše, obě tyto operace vyžadují vygenerovaný klíč. Ten vrací v požadované podobě metoda `Key::loadFromAsciiSafeString()`, do jejíhož parametru musí být předán klíč ve formě řetězce, který byl dříve uložen do konfiguračního souboru.

Před vložením do databáze je řetězec představující přístupový token zašifrován metodou `Crypto::encrypt()`. Tato metoda má dva povinné parametry. Prvním je zmíněný řetězec představující token a druhým vygenerovaný klíč. Dešifrování se provádí metodou `Crypto::decrypt()`, jejíž první parametr je onen zašifrovaný token a druhý parametr je opět klíč.

5.2.3 Využití Cronu

Z hlediska synchronizace alb a událostí je důležitou částí aplikace `CliModul`. Tento modul musí být dostupný pouze přes příkazovou řádku a slouží k provádění operací, jejichž volání bude zajišťovat Cron. Těmito operacemi jsou metody třídy `App\CliModule\Presenters\CliPresenter`.

Jak bylo uvedeno v podkapitole 4.4.7, synchronizaci je možné provádět ve dvou režimech – iniciálně a inkrementálně. I tuto skutečnost bylo nutné zohlednit v implementaci třídy `CliPresenter` a routeru:

```
$router[] = new CliRouter(["action" => "Cli:Cli:initialSync"]);
$router[] = new CliRouter(["action" => "Cli:Cli:sync"]);
$router[] = new CliRouter(["action" => "Cli:Cli:syncPhotos"]);
```

Dostupnost uvedených rout přes příkazovou řádku zajišťuje využití třídy `CliRouter`. V parametru konstruktoru této třídy je uvedeno, jakým způsobem lze daná akce modulu `CliModul` a presenteru `CliPresenter` zavolat. Akcemi jsou tyto metody zmíněného presenteru:

- `actionInitialSync()` – zajišťuje iniciální synchronizaci.
- `actionSync()` – umožňuje provádět inkrementální synchronizaci.
- `actionSyncPhotos()` – stahování fotek pro alba, u nichž byla vyžádaná synchronizace. Tato akce je od ostatních dvou oddělena proto, že jí bude nutné volat v jiném časovém intervalu.

Volání metody `actionInitialSync()` by tedy na základě první předepsané routy vyžadovalo příkaz ve tvaru:

```
php *index.php Cli:Cli:initialSync
```

Na místo hvězdičky by byla dosazena cesta k souboru `index.php`.

5.3 REST API

Implementace tohoto rozhraní poskytuje CRUD operace nad jakýmkoliv typem položek. Následující podkapitoly popisují, jak toho bylo dosaženo a jak lze toto API používat.

5.3.1 Router a zasílání požadavků

Jak bylo nastíněno v návrhu struktury aplikace (podkapitola 4.2.4), pro toto rozhraní je podobně jako pro příkazovou řádku vymezen vlastní modul – v tomto případě `ApiModul`. Tento modul obsahuje `ApiPresenter`, který zajišťuje autentizaci a autorizaci uživatele a provádění operací na základě použité HTTP metody.

Aby se však uživatel do této části aplikace dostal, bylo potřeba implementovat pro zmíněný modul a presenter vhodnou routu. Její deklaraci představuje tento kód:

```
$router[] = new Route("api/<resource [a-z]+>[/<id>]", array(
    "module" => "Api",
    "presenter" => "Api",
    "action" => "default"
));
```

Uvedený kód předepisuje formát adresy, na kterou může uživatel tohoto REST API zasílat svoje požadavky. Hodnoty dosazené za `resource` a `id` předá router presenteru coby parametry. Obecně lze tvar této adresy zapsat takto:

```
http://www.domena.cz/api/resource/id
```

Jako `resource` (neboli zdroj) je zde možné dosadit název jakékoliv tabulky, která obsahuje záznamy s konkrétními položkami. Část `/id` je nepovinná a uvádí se v případě, kdy je potřeba provést akci s jedinou položkou, která je identifikována hodnotou dosazenou za `id`. Tuto hodnotu je tedy nutné zadat vždy v kontextu s metodami `PUT` a `DELETE`, které jsou využívány k úpravě a odstranění záznamu. Při použití metody `GET` je třeba zadat `id` pouze v případě, kdy má být vrácen jediný záznam. Jinak jsou vypsány všechny položky daného typu. Poslední povolenou metodou je v tomto API metoda `POST`, u které je využití `id` vyloučené.

Konkrétní použití může být následující. Uživatel webu Police Symphony Orchestra chce prostřednictvím REST API odstranit ze systému novinku s identifikačním číslem 1. Musí tedy využít metodu `DELETE` a jako zdroj uvede název tabulky se záznamy o novinkách. Ta je pojmenovaná jako `news`. Zašle tedy tento požadavek:

```
DELETE /api/news/1 HTTP/1.1
Host: www.policesymphonyorchestra.cz
```

Výčet a povinnost parametrů, které lze vkládat metodou `POST` nebo upravovat metodou `PUT`, je ekvivalentní s výčtem a povinností atributů tabulky, nad kterou je daná operace prováděna. U položek, které mohou být otevřeny na samostatné stránce, je třeba vložit také hodnotu pro atribut `title` v tabulce `items`.

5.3.2 Modelová vrstva

Všechny zmíněné operace zajišťují třídy modelové vrstvy. Tyto třídy se musí nacházet ve jmenném prostoru `App\Model\Item` a musí mít název ve tvaru `*Model`,

kde hvězdička představuje název tabulky, nad kterou daná modelová třída provádí operace.

Dále je nutné, aby tyto třídy implementovaly společné rozhraní, které bude předepisovat metody používané třídou `App\ApiModule\Presenters\ApiPresenter`. Tímto rozhraním je `IApiModel` a požaduje implementaci těchto metod:

- `get($id)` – třída `ApiPresenter` volá tuto metodu při `GET` požadavku s uvedeným `id` parametrem. Vrací jediný záznam.
- `getList()` – opět metoda pro `GET` požadavek. Tentokrát bez uvedeného `id` parametru. Vrací všechny záznamy pro požadovaný zdroj.
- `insert($values)` – metoda volaná u `POST` požadavku. V parametru vyžaduje asociativní pole s hodnotami, které mají být vloženy jako další záznam tabulky. Klíče tohoto pole musí odpovídat názvům sloupců této tabulky.
- `update($id, $values)` – `ApiPresenter` volá tuto metodu při `PUT` požadavku. V prvním parametru je nutné předat `id` záznamu, který má být upraven a ve druhém (stejně jako u předchozí metody) pole hodnot.
- `delete($id)` – vyřizuje `DELETE` požadavek. Odstraní položku s předaným `id`.

Výběr té správné modelové třídy pro daný typ položky (resp. zdroje) obstarává metoda `setResourceModel()` ve třídě `ApiPresenter`. Následuje zjednodušený kód této metody:

```
try {
    $modelName = ucfirst($resourceName);
    $this->model = $dic->getByType(
        "App\\Model\\Item\\" . $modelName . "Model"
    );

} catch (Nette\DI\MissingServiceException $e) {
    // chyba 404 - Not Found
}
```

Na základě názvu zdroje `$resourceName` (který je předáván v parametru metody `setResourceModel()`) je tedy z dependency injection kontejneru `$dic` vyžádána služba, která je instancí požadovaného modelu. Název této služby je složen ze jmenného prostoru `App\Model\Item`, ve kterém se nachází modelové třídy umožňující operace nad položkami, názvu zdroje a slova `Model`.

Tímto způsobem by samozřejmě mohlo dojít k výběru modelu, který neexistuje (resp. není zaregistrován jako služba v DI kontejneru) nebo není určen k operacím nad položkami. Obě tyto situace vyvolají výjimku a uživateli je ohlášena chyba HTTP kódem 404.

5.3.3 Autentizace a autorizace

Zasílat požadavky metodou `GET` mohou všichni uživatelé, aniž by museli projít autorizačním procesem. API však poskytuje také operace, které umožňují zásah do databáze systému. Je tedy třeba předejít tomu, aby k nim měla přístup neautorizovaná osoba.

Samotné autorizaci předchází autentizace, která byla realizována přes PHP session. Tento způsob autentizace byl zvolen s ohledem na jednotnost přihlašování uživatelů v rámci celého systému. Uživatel požadující přístup ke všem operacím, které API poskytuje, musí postupovat tímto způsobem:

1. Uživatel zašle `POST` požadavek na `/api/login`. V těle tohoto požadavku uvede svoje přihlašovací údaje (email a heslo).
2. Pokud v systému existuje uživatelský účet s uvedenými přihlašovacími údaji, dojde k přihlášení uživatele a systém mu vrátí `PHPSESSID` řetězec.
3. Získané `PHPSESSID` bude uživatel zasílat v záhlaví požadavku coby hodnotu parametru `Cookie`. To znamená v tomto tvaru:
   `Cookie: PHPSESSID=získaný_řetězec`

Krok 1 a 2 je tedy třeba implementovat na straně redakčního systému a krok 3 v aplikaci klienta. Autentizaci na straně systému řeší metoda `authentication()` třídy `ApiPresenter`. V případech, kdy uživatel nezadá přihlašovací údaje nebo zadá údaje neplatné, je vrácen HTTP kód 401.

Poté, co je uživatel přihlášen, je nutné ověřit, jestli má potřebná práva k provádění požadovaných operací. Přiřazování přístupových práv řeší v této aplikaci třída `App\Auth\Authorizator`. Nastavení přístupových práv pro API znázorňuje tento blok kódu:

```
$this->acl->addRole("guest");
$this->acl->addRole("api_admin");
$this->acl->addRole("admin");

$this->acl->addResource("Api");

$this->acl->allow("guest", "Api", "GET");
$this->acl->allow("api_admin", "Api", Permission::ALL);
$this->acl->allow("admin", Permission::ALL, Permission::ALL);
```

Nejprve tedy bylo nutné přidat role a zdroje, pro které mají být práva nastavena. K tomu slouží metody `addRole()` a `addResource()`. Poté následuje přiřazení práv metodou `allow()`. V prvním parametru této metody je dosazena role, v dalším zdroj a v posledním operace, kterou uživatel s touto rolí může provádět v rámci definovaného zdroje. Přístupová práva definovaná výše uvedeným zdrojovým kódem lze tedy číst takto:

- `guest` – uživatelé s touto rolí mají v rámci API pouze právo na provedení operace `GET`. Jde o výchozí roli, kterou mají i nepřihlášení uživatelé.
- `api_admin` – uživatel s touto rolí má přístup ke všem operacím v rámci API.
- `admin` – role administrátora systému, který má přístup ke všem operacím.

Třída `Authorizator` implementuje rozhraní `Nette\Security\IAuthorizator`. Musí proto obsahovat metodu `isAllowed()`, která ověřuje, jestli má daná role právo provádět požadovanou operaci nad určitým zdrojem. Tuto metodu pak využívá třída `Nette\Security\User`. Ověření přístupových práv je proto ve třídě `ApiPresenter` realizováno takto:

```
if ($this->getUser()->isAllowed("Api", $method)) {
    $this->setResourceModel($resource);
} else {
    // chyba 403 - Forbidden
}
```

Metodou `getUser()` lze v presenteru získat instanci zmíněné třídy `User`. Ta reprezentuje přihlášeného uživatele. Protože je role jednou z vlastností, které si tento objekt uchovává, metoda `isAllowed()` v této třídě nevyžaduje její předání do parametru. Předává se tedy pouze zdroj, kterým je zde `Api` a proměnná `$method`, která představuje metodu požadavku.

Pokud má přihlášený uživatel právo provádět danou operaci, dojde k nastavení požadovaného modelu (viz podkapitola 5.3.2), v opačném případě je ohlášena chyba HTTP kódem 403.

6 Nasazení a údržba

Následující podkapitoly se věnují problematice nasazení a údržby instancí systému.

6.1 Nasazení

Doporučený postup při nasazování instance systému:

1. Založení nového projektu.
2. Vytvoření databáze a import dumpu databáze pro výchozí projekt.
3. Vytvoření sekcí webu a uživatelských účtů.
4. Registrace aplikace na stránkách Facebooku, Googlu a Twitteru.
5. Vytvoření nových nebo vložení a konfigurace existujících komponent.
6. Úprava layoutu[10], kaskádových stylů a javascriptových souborů.

Založením nového projektu je myšleno naklonování výchozího projektu z Git repozitáře a vytvoření nového repozitáře kam bude následně tento projekt vložen coby nová instance systému. V této práci lze výchozí projekt najít v příloze D v adresáři `default_project/project`. Tento projekt obsahuje pouze komponenty společné pro všechny instance systému (viz podkapitola 5.1.2).

Dále je nutné vytvořit databázi a importovat do ní SQL soubor, který obsahuje definici výchozích tabulek a záznamů pro redakční systém. Jde o soubor `default_project/dump.sql` (opět umístěn v příloze D). Údaje o databázi je třeba zanést do sekce `database` v konfiguračním souboru (viz podkapitola 5.1.1).

Mezi výchozími záznamy databáze je i domovská stránka v tabulce `pages` a typ stránky `page` v tabulce `pages_type`. Jde o základní typ stránky, na které je umístěn pouze formulář s WYSIWYG editorem. Pokud je na webu více typů stránky, je nutné je doplnit do tabulky `pages_type` a šablony `default.latte` presenteru `PagePresenter` (viz ukázka kódu v podkapitole 5.1.3).

Další z výchozích záznamů představuje uživatelský účet pro implementátora systému v tabulce `users`. Ten se do systému musí přihlásit zadáním emailové adresy `implementator@example.cz` a hesla `FH6qF4p6`[11]. Po zadání těchto údajů se implementátorovi zobrazí rozhraní pro správu uživatelských účtů a sekcí webu (viz obrázky v příloze E). Prostřednictvím tohoto rozhraní vytvoří uživatelský účet nebo účty a sekce webu odpovídající požadavkům koncového uživatele. Ve formuláři pro přidání sekce je nutné u každé sekce zvolit příslušný typ stránky.

Jak bylo zmíněno v podkapitole 5.2.1, pro každou instanci je nutné provést registraci aplikace na stránkách poskytovatelů. Pokud má uživatel účet na Facebooku, Twitteru nebo Googlu, který chce s danou instancí propojit, musí implementátor

[10] Layout je v šablonovacím systému Latte výchozí šablonou (Nette Foundation, 2017d.).

[11] Jde samozřejmě o přihlašovací údaje použitelné pouze v případě projektu v příloze D.

získat také ID těchto účtů. Získané údaje musí být uvedeny do sekce `parameters` v konfiguračním souboru. Podrobnosti k získávání těchto údajů jsou uvedeny v podkapitole 5.2.1.

S nasazováním instance systému je spojeno také vytváření nových komponent odpovídajících požadavkům koncového uživatele. Této problematice se věnuje podkapitola 5.1.3. Možné je samozřejmě využít i komponenty, které byly vytvořeny pro jiné instance systému.

Požadavkům uživatele je nutné přizpůsobit také vzhled webových stránek. K tomu účelu slouží soubory s kaskádovými styly a javascriptové soubory. V adresáři výchozího projektu jsou CSS soubory umístěny v podadresáři `www/css`. Vzhled dané instance systému určují především styly v souboru `main.css`. V souboru `admin.css` jsou styly týkající se administračního rozhraní.

Javascriptové soubory jsou umístěny v podadresáři `www/js` výchozího projektu. Zde je soubor `common.js`, `admin.js` a `custom.js`. První dva uvedené jsou společné pro všechny instance, přičemž `admin.js` využívají pouze administrační prvky webu. Soubor `custom.js` je určen pro definici skriptů, které se týkají pouze konkrétní instance systému.

Pokud daná instance systému vyžaduje využití javascriptové knihovny, která ve výchozím projektu není, je třeba jí vložit do bloku `scripts` v layoutu `@customLayout.latte`. Tento layout obsahuje i další bloky, které je nutné při nasazení nového systému upravit. Jde o blok `head` a `footer`. V prvním zmíněném je třeba nastavit tagy záhlaví HTML dokumentu. Druhý blok představuje zápatí webu.

6.2 Údržba

Nasazené instance systému je třeba udržovat. Nejde jen o provádění úprav v rámci každé instance zvlášť (například úpravu vzhledu). Změny realizované ve společných částech systému je z hlediska konzistence systému žádoucí provést ve všech jeho instancích.

S rostoucím počtem instancí rostou také nároky na údržbu. V této kapitole bude naznačen způsob, jakým by bylo možné provádět zmíněné plošné aktualizace instancí systému i v případě většího množství nasazených instancí.

Především je nutné od sebe oddělit do vlastních repozitářů jednotlivé komponenty a část systému, která je společná pro všechny instance. S nasazením nové instance systému by pak implementátor zadal, ze kterých repozitářů má být daná instance tvořena. Pokud by byla v některém z repozitářů provedena změna, implementátor by zadal příkaz, kterým by tuto změnu nahrál do všech instancí systému, které jsou daným repozitářem tvořeny.

Tento způsob údržby je možné realizovat například ve verzovacím systému Git s využitím tzv. *submodulů* (Git, 2017). Každá instance systému má vlastní repozitář, u kterého je možné přidat tyto submoduly. Těmi by mohly být právě repozitáře oddělených částí systémů. S každou úpravou submodulu by pak bylo nutné zadat

příkaz `git submodule update` v repozitářích všech instancí systému, které tento submodul využívají. Jestli by však byl tento způsob vhodným řešením, je třeba prověřit prakticky s větším počtem nasazených instancí systému.

7 Závěr

Cílem této práce bylo realizovat redakční systém, který nabídne snazší způsob správy webových stránek než běžně používané redakční systémy. Požadavky uvedené v kapitole 2 vyžadují především odstínění koncového uživatele od funkcí, které nejsou přímo potřebné pro úpravu obsahu webu. Dalším důležitým požadavkem je zajištění co největší přímočarosti akcí, které potřebuje koncový uživatel v systému provádět. Veškeré akce týkající se úpravy obsahu stránky by proto mělo být možné provádět přímo na stránce, které se tato úprava týká. Tyto vlastnosti nesplňoval žádný z běžně používaných redakčních systému analyzovaných v kapitole 3.

Na základě zmíněných požadavků jsem tedy navrhnul vlastní řešení. Aby systém splňoval požadavky uživatele, ale zároveň neobsahoval příliš mnoho nabídek a nastavení, je potřeba pro každého uživatele vytvořit instanci systému na míru. Proto bylo třeba zohlednit také požadavky z pohledu implementátora, který chce instance systému co nejsnáze nasazovat. Hlavní důraz byl tedy kladen na správný návrh struktury aplikace (podkapitola 4.2) a databáze (podkapitola 4.3). Uživatelem požadované funkce je možné do systému zavést vytvořením a konfigurací znovupoužitelných komponent (viz podkapitola 5.1.3).

Další požadovanou vlastností tohoto systému byla integrace webových služeb. I tato vlastnost systému cílí na usnadnění práce koncového uživatele. Navrženy a následně implementovány byly třídy pro získávání přístupových tokenů k webovým službám různých poskytovatelů (viz podkapitola 4.4.1 a 5.2.1). Tyto třídy jsou základem pro provádění operací, jako je přihlašování přes sociální sítě, vkládání statusů na sociální sítě souběžně s vložením článku na stránku, vkládání událostí na stránku a zároveň do Google kalendáře a automatizované stahování fotoalb a událostí z Facebooku. Všechny tyto funkce byly navrženy a implementovány.

Provozuschopnost tohoto systému byla otestována jeho nasazením na web Police Symphony Orchestra. Tento web je dostupný na adrese `xpospech.php5.cz`. Zdrojový kód tohoto řešení je k dispozici v příloze D. Do systému se lze přihlásit jako implementátor webu zadáním emailové adresy `implementator@example.cz` a hesla `FH6qF4p6` do formuláře, který se zobrazí po kliknutí na text *administrace* v zápatí webu.

Námětem pro další vývoj systému je především zdokonalení způsobu, jakým je aplikace členěna z hlediska šablon a kaskádových stylů. Problematikou, která bude vyžadovat komplexní řešení, je rovněž údržba systému naznačená v podkapitole 6.2.

8 Literatura

Andreev, I., 2012. *Dare to build vertical design with relational data.* SlideShare [online]. [cit. 2017-04-23]. Dostupné z: https://www.slideshare.net/ivoandreev/entity-attribute-value-14534367.

Boyd, R, 2012. *Getting started with OAuth 2.0.* Sebastopol, Calif.: O'Reilly. ISBN 1449311601.

BuiltWith, 2017. *Blog technologies Web Usage Statistics.* BuiltWith Technology Lookup [online]. [cit. 2017-05-05]. Dostupné z: https://trends.builtwith.com/cms/blog.

Buytaert, D., 2017a. *Who Uses Drupal?* Drupal.com [online]. [cit. 2017-05-15]. Dostupné z: https://drupal.com/showcases.

Buytaert, D., 2017b. *What is Drupal?* Drupal.com [online]. [cit. 2017-05-15]. Dostupné z: https://drupal.com/product/web-content-management.

Buytaert, D., 2017c. *Drupal for Commerce.* Drupal.com [online]. [cit. 2017-05-15]. Dostupné z: http://www.drupal.com/product/commerce.

Buytaert, D., 2017d. *Download & Extend.* Drupal.org [online]. [cit. 2017-05-15]. Dostupné z: https://www.drupal.org/download.

Buytaert, D., 2017e. *Social Timeline.* Drupal.org [online]. [cit. 2017-05-15]. Dostupné z: https://www.drupal.org/project/social_timeline.

Combell, 2017. *Which CMS should you choose for your website: WordPress, Joomla or Drupal?* Combell blog [online]. [cit. 2017-05-06]. Dostupné z: https://www.combell.com/en/blog/which-cms-should-you-choose-for-your-website-wordpress-joomla-or-drupal/.

Envato, 2017a. *2017's Newest Premium Wordpress Themes.* ThemeForest [online]. [cit. 2017-05-15]. Dostupné z: https://themeforest.net/category/wordpress.

Envato, 2017b. *Joomla Templates.* ThemeForest [online]. [cit. 2017-05-15]. Dostupné z: https://themeforest.net/category/cms-themes/joomla.

Facebook, 2017a. *Page Plugin.* Facebook for Developers [online]. [cit. 2017-05-15]. Dostupné z: https://developers.facebook.com/docs/plugins/page-plugin.

Facebook, 2017b. *Expiration and Extension - Facebook Login.* Facebook for Developers [online]. [cit. 2017-04-25]. Dostupné z: https://developers.facebook.com/docs/facebook-login/access-tokens/expiration-and-extension.

Foster, E., Godbole, S., 2016. *Database Systems: A Pragmatic Approach.* 2. New York: Apress. ISBN 978-1-4842-1192-2.

Git, 2017. *Documentation git-submodule.* Git [online]. [cit. 2017-05-15]. Dostupné z: `https://git-scm.com/docs/git-submodule`.

GitHub, 2017a. *Simple Encryption in PHP.* GitHub [online]. [cit. 2017-05-15]. Dostupné z: `https://github.com/defuse/php-encryption`.

GitHub, 2017b. *Encodes and decodes NEON file format.* GitHub [online]. [cit. 2017-05-15]. Dostupné z: `https://github.com/nette/neon`.

Google, 2017a. *Using OAuth 2.0 for Web Server Applications.* Google Developers [online]. [cit. 2017-04-25]. Dostupné z: `https://developers.google.com/identity/protocols/OAuth2WebServer`.

Google, 2017b. *Getting Started with the Google+ Domains API.* Google Developers [online]. [cit. 2017-05-01]. Dostupné z: `https://developers.google.com/+/domains/getting-started`.

Google, 2017c. *Developer Offerings.* Google Developers [online]. [cit. 2017-05-01]. Dostupné z: `https://developers.google.com/google-apps/products`.

Hammer, E., 2010. *Introducing OAuth 2.0.* [online]. [cit. 2017-04-26]. Dostupné z: `https://hueniverse.com/introducing-oauth-2-0-b5681da60ce2`.

Hay, D., 2013. *Data Model Patterns: Conventions of Thought.* Boston: Addison-Wesley. ISBN 9780133488623.

IETF, 2012. *RFC 6749 - The OAuth 2.0 Authorization Framework.* IETF [online]. [cit. 2017-04-26]. Dostupné z: `https://tools.ietf.org/html/rfc6749`.

Jetimpex, 2017. *8 Most Popular Websites Using Joomla.* MonsterPost [online]. [cit. 2017-05-15]. Dostupné z: `https://www.templatemonster.com/blog/8-popular-websites-using-joomla/`.

JoomlaUX, 2016. *JUX Social Gallery.* JoomlaUX [online]. [cit. 2017-05-15]. Dostupné z: `https://joomlaux.com/jux-extensions/image/social-gallery.html`.

Löper, D., Klettke, M., Bruder, I. a kol., 2013. Enabling flexible integration of healthcare information using the entity-attribute-value storage model. *Health Information Science and Systems.* [online]. 1(1) [cit. 2017-04-23]. DOI: 10.1186/2047-2501-1-9. ISSN 2047-2501. Dostupné z: `http://link.springer.com/10.1186/2047-2501-1-9`.

Nette Foundation, 2017a. *Píšeme komponenty.* Nette Framework [online]. [cit. 2017-05-15]. Dostupné z: `https://doc.nette.org/cs/2.4/components`.

Nette Foundation, 2017b. *Routování URL.* Nette Framework [online]. [cit. 2017-05-15]. Dostupné z: https://doc.nette.org/cs/2.4/routing.

Nette Foundation, 2017c. *Slovníček pojmů.* Nette Framework [online]. [cit. 2017-05-15]. Dostupné z: https://doc.nette.org/cs/0.9/slovnik.

Nette Foundation, 2017d. *Výchozí Latte makra.* Nette Framework [online]. [cit. 2017-05-15]. Dostupné z: https://latte.nette.org/cs/macros.

Open Source Matters, 2017a. *About Joomla!.* Joomla! [online]. [cit. 2017-05-15]. Dostupné z: https://www.joomla.org/about-joomla.html.

Open Source Matters, 2017b. *Joomla! Core Features.* Joomla! [online]. [cit. 2017-05-15]. Dostupné z: https://www.joomla.org/core-features.html.

Open Source Matters, 2017c. *Joomla 3.0 version history.* Joomla! Documentation [online]. [cit. 2017-05-15]. Dostupné z: https://docs.joomla.org/Joomla_3.0_version_history.

Oracle Corporation, 2017. *Encryption and Compression Functions.* MySQL [online]. [cit. 2017-04-26]. Dostupné z: https://dev.mysql.com/doc/refman/5.5/en/encryption-functions.html.

Q-Success, 2017. *Usage of content management systems for websites.* W3Techs [online]. [cit. 2017-05-05]. Dostupné z: https://w3techs.com/technologies/overview/content_management/all.

Srizon, 2017. *Srizon Facebook Album.* Srizon [online]. [cit. 2017-05-15]. Dostupné z: https://www.srizon.com/srizon-facebook-album.

Twitter, 2017a. *Application-only authentication.* Twitter Developers [online]. [cit. 2017-04-25]. Dostupné z: https://dev.twitter.com/oauth/application-only.

Twitter, 2017b. *OAuth FAQ.* Twitter Developers [online]. [cit. 2017-04-25]. Dostupné z: https://dev.twitter.com/oauth/overview/faq.

Twitter, 2017c. *Single-user OAuth with Examples.* Twitter Developers [online]. [cit. 2017-04-25]. Dostupné z: https://dev.twitter.com/oauth/overview/single-user.

Walls, C., Donald, K., Clarkson, R., 2014. *Spring Social Reference.* Spring [online]. [cit. 2017-04-25]. Dostupné z: http://docs.spring.io/spring-social/docs/1.1.0.RELEASE/reference/htmlsingle.

WebKul, 2017. *Joomla Latest Tweet.* WebKul Store [online]. [cit. 2017-05-15]. Dostupné z: https://store.webkul.com/Joomla-Extensions/Latest-Tweet.html.

Wikimedia Foundation, 2017. *Semantic URL.* Wikipedia [online]. [cit. 2017-05-15]. Dostupné z: https://en.wikipedia.org/wiki/Semantic_URL.

Wordpress, 2017a. *Our Clients.* WordPress.com VIP [online]. [cit. 2017-05-05]. Dostupné z: https://vip.wordpress.com/clients/.

Wordpress, 2017b. *Our Services.* WordPress.com VIP [online]. [cit. 2017-05-05]. Dostupné z: https://vip.wordpress.com/our-services/.

Wordpress, 2017c. *About WordPress.* WordPress.org [online]. [cit. 2017-05-05]. Dostupné z: https://wordpress.org/about/.

Wordpress, 2017d. *Features.* WordPress.org [online]. [cit. 2017-05-05]. Dostupné z: https://wordpress.org/about/features/.

Wordpress, 2017e. *Custom Facebook Feed.* Wordpress.org [online]. [cit. 2017-05-15]. Dostupné z: https://wordpress.org/plugins/custom-facebook-feed/.

Wordpress, 2017f. *Custom Twitter Feeds.* Wordpress.org [online]. [cit. 2017-05-15]. Dostupné z: https://wordpress.org/plugins/custom-twitter-feeds/.

Wordpress, 2017g. *Instagram Feed WD – Instagram Gallery for WordPress.* Wordpress.org [online]. [cit. 2017-05-15]. Dostupné z: https://wordpress.org/plugins/wd-instagram-feed/.

Wordpress, 2017h. *NextScripts: Social Networks Auto-Poster.* Wordpress.org [online]. [cit. 2017-05-15]. Dostupné z: https://cs.wordpress.org/plugins/social-networks-auto-poster-facebook-twitter-g/.

Wordpress, 2017i. *Simple Calendar – Google Calendar Plugin.* Wordpress.org [online]. [cit. 2017-05-15]. Dostupné z: https://wordpress.org/plugins/google-calendar-events/.

Wordpress, 2017j. *Photonic Gallery for Flickr, Picasa, SmugMug, 500px, Zenfolio and Instagram.* Wordpress.org [online]. [cit. 2017-05-15]. Dostupné z: https://wordpress.org/plugins/photonic/.

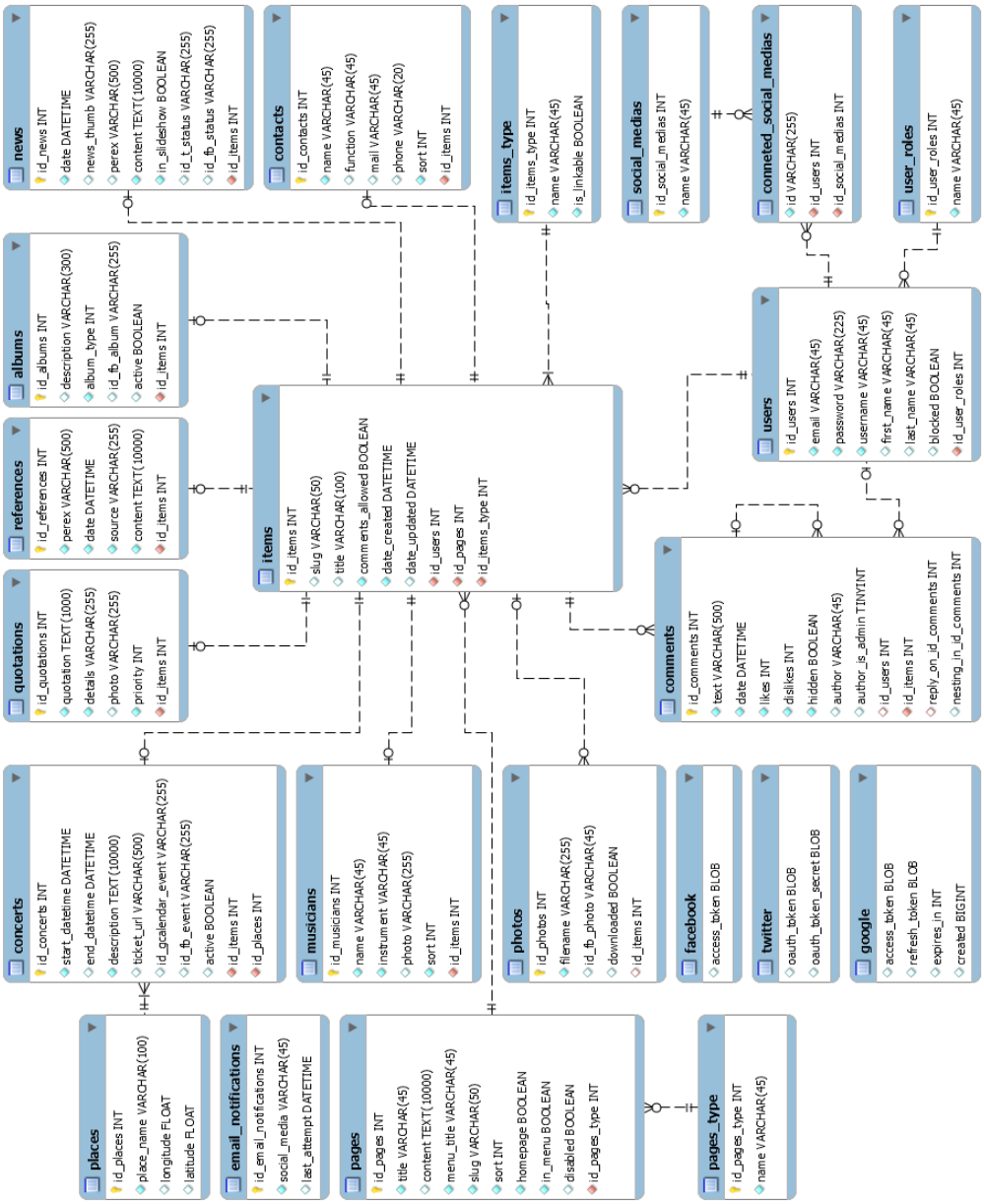
Xylus Themes, 2017. *Import Facebook Events.* Xylus Themes [online]. [cit. 2017-05-15]. Dostupné z: https://xylusthemes.com/plugins/import-facebook-events/.

Přílohy

A Ukázka rozvržení webu

Obrázek 21: Wireframe znázorňující rozdíl mezi sekcí a položkou webu.

B ERD databáze



Obrázek 22: ER diagram databáze webu Police Symphony Orchestra.

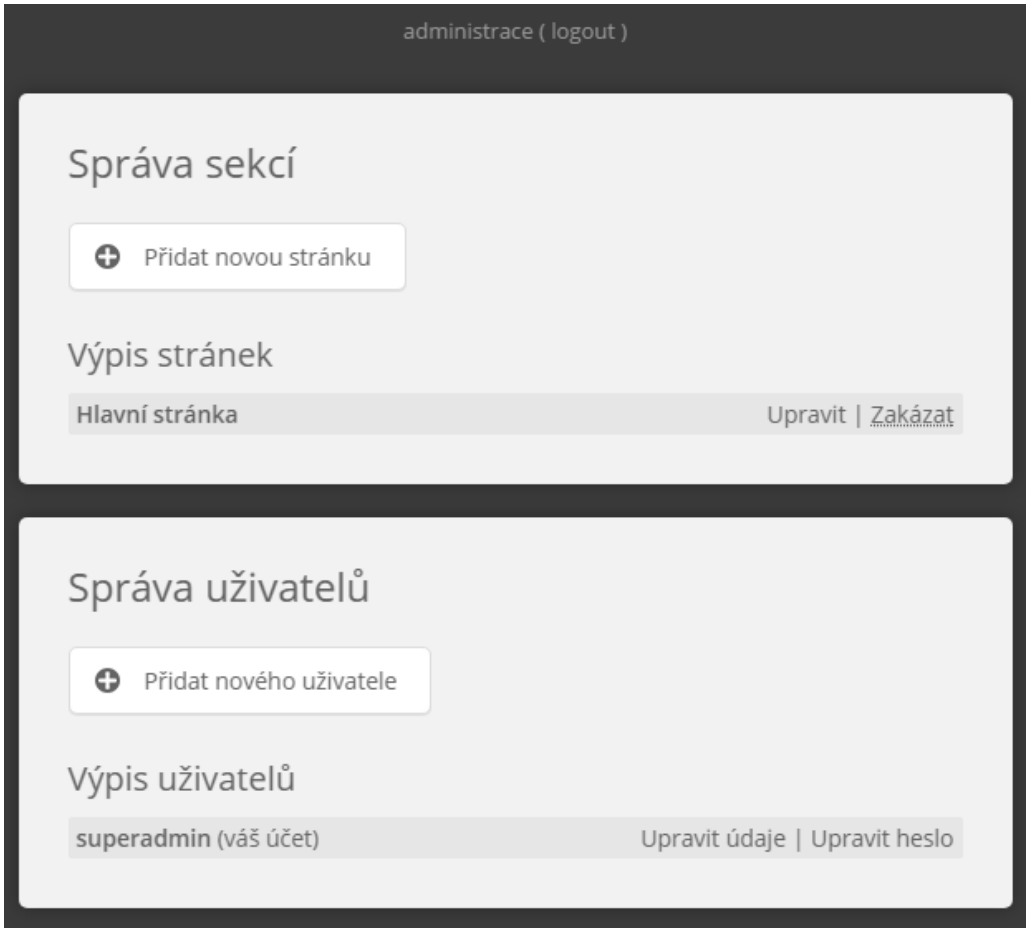
C Ukázka konfiguračního souboru

```
parameters:
    contactEmails:
        supportMail: 'support@example.com'
        adminMail: 'admin@example.com'
    facebook:
        pageId: '1387327667997045'
        apiId: '221978694946369'
        secret: '689cf0a6a5a9475c96ce487dx6921ce3'
        services:
            albums:
                active: true
                automatic: false
                downloadLimit: 180
            events:
                active: true
                automatic: false
            status:
                active: false

services:
    - App\SocialMedia\Auth\FacebookPageHandler(%facebook%)

database:
    dsn: 'mysql:host=127.0.0.1;dbname=pso'
    user: root
    password:
    options:
        lazy: yes

php:
    date.timezone: Europe/Prague

application:
    errorPresenter: Front:Error
    mapping:
        *: App\*Module\Presenters\*Presenter

session:
    expiration: 14 days
    autoStart: yes
```

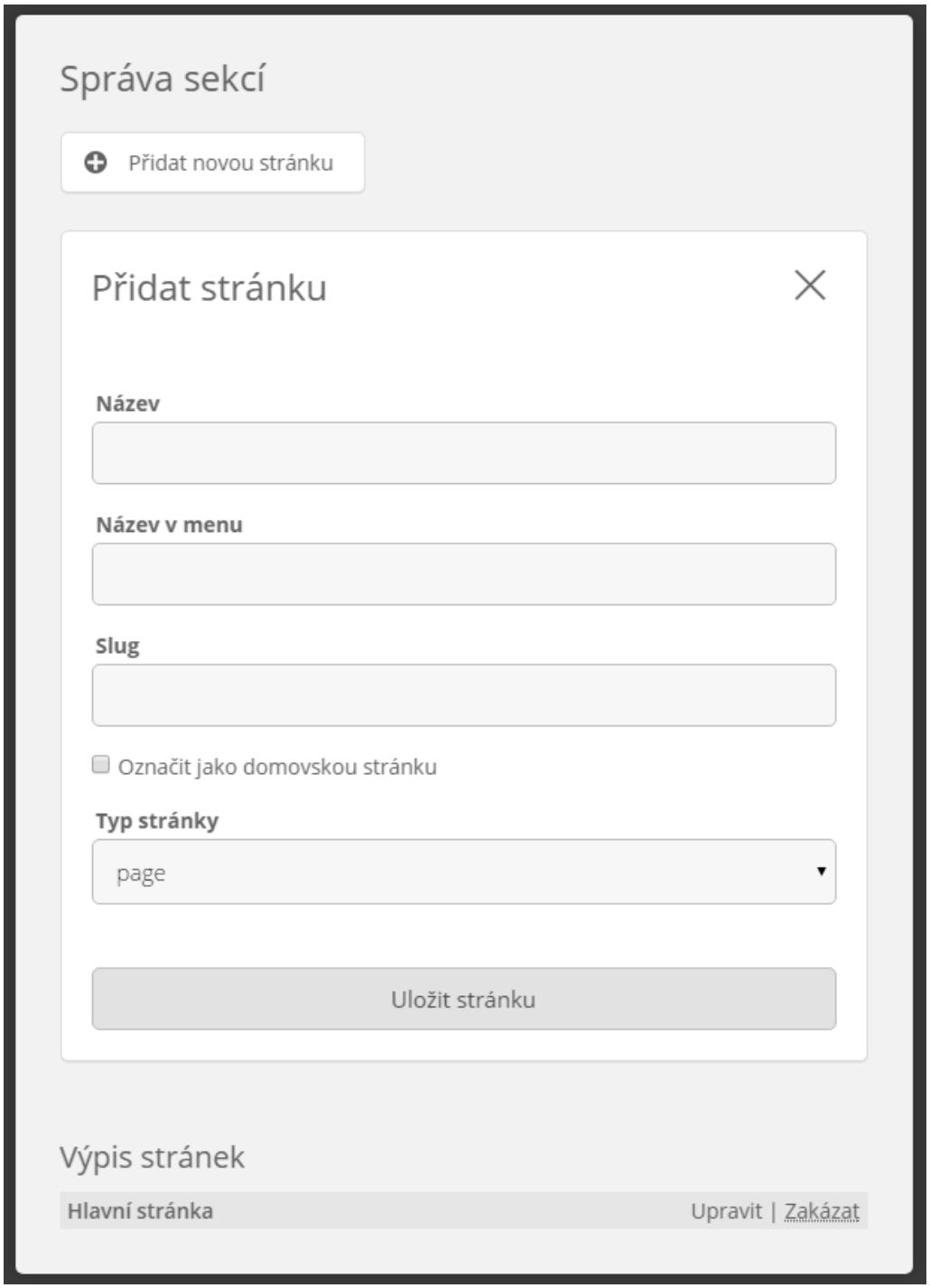
D CD

Na přiloženém CD je umístěna elektronická verze této práce ve formátu PDF a dva adresáře. Jedním z nich je adresář `default_project` s výchozím projektem, který bude využíván při nasazování nových instancí systému. Ve druhém adresáři, kterým je `pso_project`, se nachází konkrétní instance systému vytvořená podle požadavků orchestru Police Symphony Orchestra. V obou zmíněných adresářích je složka s Nette projektem a dump a schéma databáze.

E Administrační rozhraní pro implementátora



Obrázek 23: Ukázka administračního rozhraní pro implementátora webu.



Obrázek 24: Ukázka formuláře pro přidání sekce webu.